BOLOTWEET 2.0

Álvaro Ortego Marcos

TRABAJO FIN DE GRADO EN INGENIERÍA DEL SOFTWARE

Curso 2013-2014

Madrid, junio de 2014

***Director*: Jorge Jesús Gómez Sanz**

Departamento de Ingeniería del Software e Inteligencia Artificial

Facultad de Informática

Universidad Complutense de Madrid

Autorización de difusión y utilización

El abajo firmante, matriculado en el Grado de Ingeniería del Software impartido por la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo de Fin de Grado: *BoloTweet 2.0*, realizado durante el curso académico 2013-2014 y bajo la dirección de Jorge Jesús Gómez Sanz en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet, y garantizar su preservación y acceso a largo plazo.

Álvaro Ortego Marcos

Madrid, junio de 2014

Antes de nada me gustaría dedicar unas pequeñas líneas a todas aquellas personas que han hecho que todo esto sea posible.

En primer lugar dar las gracias a mi familia, en especial a mis padres y hermana, por haberme apoyado y haber confiado en mí como lo han hecho todos estos años. Sin ellos, este trabajo, esta carrera, y esta nueva etapa no habría sido posible.

Por otro lado, quisiera agradecer a todos mis compañeros, a mis amigos, por haber compartido conmigo esta experiencia, y por haberme regalado momentos inolvidables.

Por último, me gustaría agradecer todo el apoyo y la confianza que he recibido de una persona muy especial, Raquel.

GRACIAS a todos por confiar en mí.

Agradecimientos

Me gustaría agradecer personalmente a las personas que han querido colaborar y aportar en este proyecto, en especial a Jorge Jesús Gómez Sanz, y a los profesores Juan Pavón, Carlos Cervigon, Francisco Garijo y Javier Arroyo, por querer unirse y ayudar a que este trabajo salga adelante.

Gracias al departamento de Ingeniería del Software e Inteligencia Artificial de la UCM por acogerme y dejarme colaborar con ellos. Reconocer también al grupo *GRASIA*, por su implicación en el proyecto, y por proporcionar recursos para que *BoloTweet* se haya podido llevar a cabo, especialmente a Jorge Gómez Sanz, por confiar en mí desde el principio, y apoyarme a lo largo de todo este curso.

Quisiera también mostrar mi agradecimiento a la Universidad Complutense de Madrid, por haber aprobado y concedido el título de PIMCD a este proyecto, un reconocimiento muy importante al trabajo realizado y que le proporciona un mayor alcance.

Mencionar y agradecer al Ministerio de Educación, Cultura y Deportes, por la concesión de una Beca de Colaboración en Departamento para el curso 2013/2014, que me ha permitido una mayor dedicación e investigación sobre el proyecto.

Por último, una mención especial a la plataforma *Status.Net*, por proporcionar su software de *micro-blogging* de manera libre y abierta, bajo Licencia *Creative Commons*.

Índice

Apéndices

Índice de ilustraciones

Índice de abreviaturas

Ajax	*Asynchronous JavaScript And XML*
Bash	*Bourne again Shell*
CSS	*Cascading Style Sheets*
CSV	*Comma-separated Values*
CV	*Campus Virtual*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
INE	*Instituto Nacional de Estadística*
OTA	*Over the air*
PHP	*Hypertext Pre-processor*
PIMCD	*Proyecto de Innovación y Mejora de la Calidad Docente*
PK	*Primary Key*
SCM	*Source Code Management*
SSL	*Secure Sockets Layer*
TAM	*Technology Acceptance Model*
TIC	*Tecnologías de la Información y la comunicación*
UCM	*Universidad Complutense de Madrid*
URL	*Uniform Resource Locator*

Resumen

Los nuevos planes de estudio, cada vez apuestan más por una enseñanza personalizada y centrada en el alumno. Este hecho, conlleva un mayor esfuerzo por parte de los profesores. Aprovechando que en la sociedad actual las redes sociales son cada vez más comunes, se ha decidido estudiar y apostar por un nuevo modelo de ayuda a la docencia, basado en el uso de este tipo de herramientas, concretamente, las basadas en micro-anotaciones o micro-*blogging*. Este proyecto se plantea como una continuación a *BoloTweet*, un sistema que ofrece una metodología basada en micro-anotaciones y desarrollado por la Universidad Complutense de Madrid como proyecto PIMCD durante dos años. El proyecto *BoloTweet*, en su último año, obtuvo evidencias que señalaban una alta correlación entre las notas obtenidas en las asignaturas y las obtenidas en *BoloTweet*. Por ello, se ha continuado este trabajo en esta Memoria de Trabajo de Fin de Grado.

A lo largo de este trabajo, se expondrán las mejoras realizadas al sistema *BoloTweet* para poder soportar un método docente innovador y asumible. Algunas de estas mejoras son debidas a actualizaciones del software en que se basa este sistema, pero otras son contribuciones originales que aportan nueva funcionalidad. Tanto unas como otras han sido evaluadas en este trabajo de forma crítica, usando encuestas para el alumnado y revisión de estadísticas de uso.

Palabras clave: Metodología Docente, Microanotaciones, Evaluación Temprana, Evaluación continua, Microblogueo, e-learning, Redes Sociales, Bolonia.

Abstract

The new study plans are increasingly favoring an individual and student-centered teaching model. On the one hand, teachers have to invest more effort in this model. On the other hand, there is an increase of the use of ICT solutions in current society that could reduce such effort, specially social networks and, in particular, micro-annotacion/micro-blogging alternatives. The project *BoloTweet* tried to use this trendin favor of teachers by proposing concrete applications of micro-blogging practices. This was studied along two innovation in education projects and two years in total. The final year of this proyect obtained evidences that supported a high correlation among students with high grades in *BoloTweet* and students with high grades in the subject.

This work continues this line of thinking supporting some observed flaws in the original BoloTweet. Some improvements are due to the advances in the underlying framework, *Status.Net*. Others are original contributions of this work. Both have been evaluated critically along some months by the students.

Keywords: Teaching method, Micro-annotations, Early Evaluation, Continuous Evaluation, Micro-blogging, e-learning, social networking, Bologna.

1 Introducción

En los últimos años, el sistema educativo está implantando planes de estudio centrados cada vez más en el alumno [1]. Esto conlleva un cambio del método de enseñanza, dejando a un lado las clases en las que se transmiten contenidos, el alumno los recibe de forma pasiva, y posteriormente los utiliza para aprobar los exámenes. Ahora, se propone una enseñanza en la que se forme a los alumnos con las competencias necesarias, para que sean ellos mismos quienes posteriormente investiguen, practiquen, y adquieran un conocimiento a largo plazo, lo que se conoce como *long-life-learning* [2].

Con el actual plan Bolonia [19], se incrementa el seguimiento de la clase, la realización de apuntes específicos, la propuesta de seminarios, prácticas, ejercicios, o cualquier otra herramienta que sirva de ayuda al alumno. Además, el profesor monitoriza y evalúa todo el trabajo del estudiante de forma continua [4], con el incremento en dedicación que esto conlleva. Para los profesores, en la mayoría de los casos, esto supone convertirse en *facilitadores* del trabajo del estudiante [3].

En este contexto, es de interés una solución basada en el uso de algún tipo de tecnología [5] que ayude a automatizar algunas de estas tareas. De esta manera, se conseguiría reducir el esfuerzo por parte del profesor y se podría tener un mayor control sobre los alumnos.

Las comunidades de aprendizaje colaborativo se convierten en un instrumento central, donde estudiantes aportan y trabajan en contenidos junto a otros estudiantes con un nivel de experiencia similar. Se ha demostrado que este tipo de aprendizaje en equipo es una herramienta muy fructífera para una correcta adquisición de conocimientos [6].

El concepto de una enseñanza centrada en el alumno, y su enfoque en comunidades de aprendizaje colaborativo, hace particularmente cómodo el uso de las tecnologías web 2.0 [8, 12]. Con el uso de estas herramientas, se plantea un modelo en el que los usuarios trabajan y aportan la mayor parte del contenido, mientras que el profesor no es más que un administrador de dicho trabajo.

Ahora bien, pese al interés de estas tecnologías, es necesario valorar su penetración en la sociedad y su demanda. Según datos del Instituto Nacional de Estadística [28] en el año 2013 un 97.5% de los estudiantes utilizaba al menos una vez por semana Internet, un 98.7% hacían uso de un ordenador, y un 98.3% utilizaba un teléfono móvil, todo ello durante un período de 3 meses.

Además, con el crecimiento de las TIC se ha producido un aumento en el uso de las redes sociales [18]. En el año 2013, según el INE, "*el 64,1% de los usuarios de Internet en los últimos tres meses participa en redes sociales de carácter general, como Facebook, Twitter o Tuenti, creando un perfil de usuario o enviando mensajes u otras contribuciones. Los más participativos son los estudiantes (94,8%) y los jóvenes de 16 a 24 años (94.5%).*" Según *Comscore* [29] el mayor incremento se produjo en *Twitter*, alcanzando los 5.6 millones de usuarios a finales de 2012.

Twitter, es una red social basada en un sistema de *micro-blogging* [9]. Este tipo de herramientas, permiten una comunicación a gran escala basada en micro-anotaciones, normalmente con un límite de 140 caracteres. Las anotaciones, también conocidas como *tweets*, se relacionan a través de etiquetas (*tags*). De esta manera, es posible reunir y agrupar distintos tipos de mensajes con un contenido similar. Por la extensión de las anotaciones, se consigue difundir un mensaje rápido entre una gran cantidad de personas con un mínimo esfuerzo.

Es el éxito y la expansión de *Twitter* las que se toman como argumento para explorar el micro-*blogging* en el ámbito docente. Por ello, se concibe *BoloTweet*. Esta herramienta, utiliza un sistema de micro-*blogging* como apoyo a la docencia, permitiendo un seguimiento continuo del estudiante con un mínimo esfuerzo por parte del profesor. En esta herramienta, los estudiantes participan y contribuyen con conocimiento, como en las comunidades de aprendizaje colaborativo.

BoloTweet surge como un trabajo de investigación, dirigido por Jorge Jesús Gómez Sanz, profesor de la Facultad de Informática de la Universidad Complutense de Madrid, sobre el uso de redes de *micro-blogging* en la enseñanza. *BoloTweet* se construye sobre un software social de código abierto, *StatusNet*. Este sistema, desarrollado en PHP, ofrece una plataforma de micro-*blogging* abierta, con un potencial similar a *Twitter*.

El método docente que propone *BoloTweet*, consiste en la expresión de una idea por parte de los alumnos derivada del contenido impartido en una clase. Las clases duran 50', 40' de los cuales se dedican a impartir teoría y 10' para seguimiento con *BoloTweet*. Durante estos 10 últimos minutos, los estudiantes deben enviar un tweet que resuma alguna de las ideas dichas en clase. Lo debe hacer en 140 caracteres y usando una gramática y ortografía correctas [26, 27].

La *Ilustración 1* muestra la apariencia inicial de la interfaz de usuario en *BoloTweet*, al que llamaremos *BoloTweet 1.0* para distinguirlo de la evolución que presenta este trabajo, que se llamará *BoloTweet 2.0*. Está dividida en 4 pestañas principales. La primera, muestra la línea temporal pública, común a todos los usuarios. La segunda muestra los grupos existentes. La tercera y la cuarta muestran las etiquetas utilizadas más recientes, y las anotaciones más populares.

Ilustración 1 - Interfaz de Usuario de BoloTweet 1.0

En la adaptación de esta herramienta para su uso en la docencia, se desarrolla el plugin *Grades*. Este plugin añade funcionalidad al sistema, permitiendo puntuar al profesor las anotaciones (*Ilustración 2*), ofreciendo un *feedback* rápido y directo, y por otro lado, generar listados de alumnos junto con sus calificaciones.

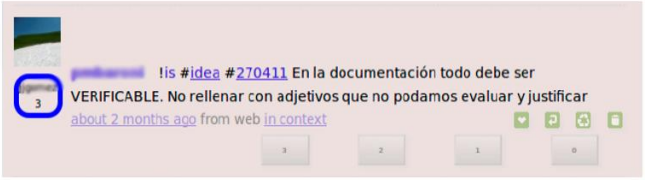


Ilustración 2 - Anotación con Grades en BoloTweet 1.0

Tras las primeras pruebas de uso de *BoloTweet 1.0* con usuarios reales, los resultados evidencian una correlación entre estudiantes que aprueban y los estudiantes con notas altas en *Bolotweet 1.0* [25].

Si bien los resultados son alentadores, la experiencia de uso no lo es tanto. En [25] se señala que hay una percepción del esfuerzo que implica usar el sistema, y que había que trabajar más en mostrar al estudiante los beneficios de usar estas micro-anotaciones. Aparte de esta deficiencia, y aunque no se mencionan en artículos citados sobre *BoloTweet 1.0,* hay otros aspectos a tener en cuenta:

- La gestión del sistema es muy deficiente. Se basaba en el procesamiento manual de listas de estudiantes para darles de alta uno a uno, por ejemplo.
- Además, no contemplaba la posibilidad de recibir ayuda en la evaluación de *tweets*. Es normal que en grupos numerosos haya un profesor responsable de la asignatura que tenga profesores de apoyo. Por otro lado, en *BoloTweet 1.0,* no había un control sobre quién puede valorar qué *tweets*. La casuística no es trivial para estos casos, pero se trata de algo importante a mejorar si el objetivo es reducir el esfuerzo de cada profesor individual.
- La privacidad es otro aspecto que no era tenido en cuenta en la versión 1.0 de *BoloTweet*. Todos los estudiantes veían las puntuaciones del resto de alumnos, además de la puntuación agregada que resulta de contabilizar todos los *tweets* de un estudiante. Esto podría llegar a dar problemas.

Por todo ello, se plantea un trabajo de fin de grado donde se exploran soluciones a estos problemas, y se proponen objetivos para hacer de *BoloTweet* una herramienta más eficaz y soportada por una metodología docente más robusta.

1.1 Objetivos y plan de trabajo

Los objetivos del presente trabajo se dividen fundamentalmente en dos, uno metodológico y uno tecnológico.

La parte metodológica se refiere a la forma en que se usa en clase *BoloTweet 2.0* para apoyo de la actividad docente. Se parte del método de apoyo a la docencia de *Bolotweet 1.0,* el cual es adaptado a los medios proporcionados por la nueva evolución. Para su ensayo, se dispone de la colaboración de varias asignaturas de la Facultad de Informática de la UCM. A través de este proceso, se puede producir una reformulación de la metodología de trabajo, o bien la validación de la misma. En este objetivo, se trabaja estrechamente con la escalabilidad y el alcance del sistema, con más de un centenar de usuarios.

En cuanto al objetivo tecnológico, se divide fundamentalmente en dos aspectos. Por un lado, tratar de manera prioritaria todas las deficiencias observadas en la versión anterior. Por otro lado, trabajar en el sentido inicial de esta herramienta, es decir, en la disminución del esfuerzo y la simplificación del trabajo docente, logrando así un menor consumo de tiempo tanto para el profesor, como para el alumno.

Es importante destacar que el objetivo principal del uso de esta herramienta consiste en maximizar el impacto, minimizando el esfuerzo. Por ello, se proponen las mejoras siguientes que serán desarrolladas en el resto de la memoria:

- Mecanismo de seguimiento del ritmo de clase: Listados de calificaciones, o informes que indiquen el estado de las anotaciones relativas a una clase concreta.

- Mejora de la privacidad: Profesores con contexto, listados privados y con puntuaciones visibles únicamente para profesores.
- Sistema de anotaciones: Mecanismo de tareas que permita automatizar la creación de anotaciones con un mínimo esfuerzo, y reduciendo lo máximo posible las equivocaciones tipográficas.
- Integración de la herramienta con el Campus Virtual: Exportación de listados de puntuaciones en formatos aceptados por la plataforma *Moodle*[30].
- Apuntes: Generación de algún tipo de mecanismo de aprovechamiento de las ideas de los propios alumnos.
- Mejora del sistema de calificación: Añadir un comportamiento renovado que favorezca la comodidad de uso por parte de los profesores.
- Herramientas de administración: Generar *scripts* que permitan una gestión del sistema por parte del administrador de una manera sencilla.

La forma de conseguir estos objetivos ha consistido en un desarrollo progresivo de las mejoras guiado por *issues* y evaluado de forma continua por los participantes. *BoloTweet* se instalaba en un entorno de desarrollo y otro de producción. En el de desarrollo se ensayaban las ideas y se probaban las mejoras. Una vez aprobadas, se subían a producción, donde eran evaluadas en condiciones reales, y desde dónde se marcaban los objetivos para el siguiente despliegue. Estos desarrollos tenían una validación final en forma de encuestas de satisfacción realizadas por los estudiantes.

1.2 Estructura de la memoria

El trabajo ha sido estructurado en 6 capítulos y 4 apéndices con la estructura que se comenta a continuación.

El capítulo 2 introduce el estado del arte para las redes sociales basadas en micro-*blogging* y orientadas a la docencia. En primer lugar, se comienza explicando este tipo de redes, así como su uso y sus principales características y funcionalidades. A continuación, se relaciona la utilización de redes de micro-*blogging* con la educación, aportando una primera visión acerca de las ventajas e inconvenientes que estas presentan. Seguidamente, se proporciona información sobre antecedentes en el uso de estos sistemas orientados al ámbito educativo, así como los resultados obtenidos. Como complemento a estos, se plantea *BoloTweet 2.0*. Por último, se muestra un estudio en forma de comparativa entre las plataformas de micro-*blogging* actuales más relevantes, y que mejor pueden adaptarse al comportamiento que se quiere proporcionar con *BoloTweet 2.0*.

El capítulo3 presenta la propuesta de *BoloTweet 2.0*. En primer lugar se describe el objetivo y el planteamiento principal de la herramienta. A continuación se detalla el método de apoyo a la docencia planteado para el uso del sistema. Se detallan una serie de directrices generales, y personalizadas tanto para alumnos como para profesores.

El capítulo 4 presenta el desarrollo llevado a cabo en *BoloTweet 2.0*. En primer lugar se describen los casos de uso incluidos en *BoloTweet 2.0*, tanto los ofrecidos por los plugins desarrollados, como por las herramientas de administración. A continuación se explica brevemente la arquitectura y el funcionamiento de la plataforma *Status.Net* y del sistema de *hooks* propuesto por la misma. Posteriormente se muestra el diseño de todos los elementos incluidos en *BoloTweet 2.0* con su consecuente explicación detallada. Seguidamente se explica la implementación y el despliegue llevado a cabo en *BoloTweet 2.0*, y se finaliza enumerando las aportaciones adicionales a los plugins realizadas sobre *Status.Net*, divididas en *Bugs* y en *Mejoras de* código.

El capítulo 5 muestra la evaluación realizada sobre el proyecto. En primer lugar se detalla la valoración del esfuerzo realizado, utilizando métricas de ingeniería del software. A continuación se expone una valoración de uso real del sistema, y se finaliza mostrando la valoración proporcionada por los estudiantes a través de encuestas.

El capítulo 6 muestra las principales conclusiones del trabajo, así como el cumplimiento de los objetivos planteados al comienzo del desarrollo. Finaliza proponiendo líneas de trabajo futuro y mejoras sobre algunos defectos encontrados durante el uso de *BoloTweet 2.0*. Al final de esta sección, se detalla la aceptación de *Bolotweet 2.0* como *PIMCD*.

El Apéndice A corresponde con el Manual de Uso del sistema a nivel de Usuario. En este manual se mencionan las funcionalidades básicas de *BoloTweet*, así como su correcto uso. Por otro lado, se detalla el método de ayuda a la docencia propuesto, y por último se explican las nuevas funcionalidades incorporadas a Bolotweet.

El Apéndice B contiene el Manual de Uso del sistema a nivel de Profesor. En este manual se detallan todas las funcionalidades incorporadas y disponibles para profesores, así como su correcto uso.

El Apéndice C contiene el Manual de Uso del sistema a nivel de Administrador. En este manual se presentan todas las herramientas administrativas desarrolladas y necesarias para una correcta administración del sistema, así como su uso detallado.

Por último, el Apéndice D recoge el diario elaborado durante todo el proceso de desarrollo. En este documento, elaborado para cumplir los requisitos de la Beca de Colaboración, se detallan todas las tareas, modificaciones, y funcionalidades añadidas a *BoloTweet*2.0 de manera detallada.

2 Estado del arte

En este capítulo se estudia el concepto de micro-*blogging* y su aplicación a la docencia. Para ello, se realiza una pequeña introducción sobre este tipo de red social, y posteriormente, se estudian otras iniciativas que utilizan o han utilizado el micro-*blogging* en distintos ámbitos, entre los cuales se quiere destacar el educativo. Se revisan las tecnologías actuales más relevantes que aplican este tipo de red social, y se propone una comparativa entre las plataformas existentes que ofrecen este tipo de tecnologías.

Las posibilidades del micro-*blogging* fueron las que motivaron la creación de *Bolotweet 1.0*, en cuya experiencia se construye *Bolotweet 2.0*, que es el objeto de esta memoria.

2.1 Introducción al micro-blogging

El micro-*blogging*, es un tipo de red social basado en anotaciones de 140 caracteres. Este tipo de sistema promueve una comunicación en tiempo real entre usuarios utilizando diferentes dispositivos, tecnologías, y aplicaciones [16].

Además, favorece una gran interacción, en la que usuarios participan y son conscientes de la actividad de otras personas de manera prácticamente instantánea [15]. Las anotaciones pueden llevar etiquetas, también conocidas como *tags*. Estos elementos no están predefinidos, pudiendo crearse tantos como se deseen, y versando sobre cualquier tema que se considere relevante. Estas etiquetas permiten encuadrar una anotación en un determinado contexto, de manera que es posible revisar y agrupar de forma sencilla un gran número de ideas. Esto favorece en gran medida el análisis del contenido [31], pudiendo conocer los *tags* más utilizados en un determinado momento, o pudiendo analizar cuáles son los temas más interesantes para un conjunto de personas. Los usuarios pueden marcar anotaciones de su interés como favoritas, incrementando su popularidad, y pudiendo posteriormente revisarlas. También es posible aumentar la fama de una anotación a través de "*retweets*", o "reenvíos" de la misma. Cada reenvío hace llegar la anotación a todos los seguidores del usuario. De esta manera, ese mensaje cada vez tendrá un mayor alcance, siendo visible por una mayor cantidad de usuarios. Con este mecanismo de publicación de anotaciones, el micro-*blogging* genera una gran cantidad de información con un coste y un esfuerzo relativamente pequeño.

Los sistemas de micro-*blogging* más conocidos actualmente son *Twitter*, *Tumblr* y *Pownce*.

El mayor ejemplo de uso lo proporciona *Twitter* [15], la red de micro-*blogging* más utilizada en el mundo. Según sus propios datos [33], disponen de unos 255 millones de usuarios activos al mes y más de 500 millones de tweets al día. En el experimento realizado en [9], se obtuvo que los usuarios de *Twitter* envían una media de 5-30 tweets a la semana.

Sobre la plataforma *Twitter* se han montado sistemas de apoyo en la educación, como se verá a continuación. Sin embargo, no está claro que estos sistemas defiendan o tengan en cuenta la privacidad del estudiante, puesto que sus usuarios admiten unas condiciones de uso particulares que no tienen que ser las que más convengan en un entorno educativo.

2.2 Micro-blogueo y educación

Las características ofrecidas por las redes sociales basadas en micro-*blogging*, con anotaciones de tan solo 140 caracteres [17], y los resultados obtenidos por algunas de ellas, como *Twitter*, han despertado un enorme interés en los investigadores para crear aplicaciones de este tipo orientadas a la educación.

Uno de los problemas principales de los alumnos es la carencia de ganas, o de motivación para realizar un trabajo si no es a cambio de algún tipo de recompensa, ya sea en forma de puntuación, o de algún otro tipo [26, 12]. Este problema se trata de subsanar en forma de red social, esto es, creando redes de usuarios donde interesa que otros conozcan lo que yo hago y viceversa. Por ejemplo, constituir comunidades en las que los usuarios pueden compartir información personal, sobre sus gustos, intereses, u otros. Esto propicia una mayor actividad sobre el sistema [11]. El trabajo en [32] es un ejemplo de esto. Los autores utilizan la plataforma de micro-*blogging Cirip.eu* para contribuir a mejorar los procesos educativos a través de métodos de enseñanza cercanos a la manera de pensar de los estudiantes. Como conclusiones, destacan las positivas consecuencias que tiene el uso de este tipo de herramientas en la enseñanza, en relación con la educación clásica, entre ellas una mayor participación, y sobre todo, una colaboración activa entre los propios estudiantes.

Otro ejemplo es [13], donde los autores explican su experiencia con *Twitter* en la enseñanza. Destacan la capacidad del micro-*blogging* para generar un mecanismo en el que estudiantes intercambian opiniones, consejos, enlaces a recursos externos, fuentes de información, por citar algunos, generando una fuente colaborativa de conocimiento.

Se plantea además como un mecanismo a través del cual, los propios alumnos puedan leer y recibir conocimiento a través de las anotaciones de sus compañeros, permitiendo así obtener diferentes puntos de vista sobre un mismo tema, o diferentes ideas sobre una misma clase [26].

En ocasiones, los propios estudiantes corrigen a otros estudiantes, consiguiendo un enfoque *student-teach-other-students* [7], en el que se disminuye esfuerzo por parte de los profesores.

Además, la limitación de caracteres provoca que el alumno se esfuerce por generar una idea, o redactar una pregunta de manera clara y concisa [21, 31].

Relacionando las funcionalidades que ofrece el *micro-blogging* con la docencia, una de las características más importantes es la utilización de etiquetas o *hashtags*. A través de ellas, es posible relacionar anotaciones de un mismo día, o de una misma materia, generando así ideas con una cierta estructura, y a las que el usuario puede acudir para revisar un aspecto concreto [26]. Cada etiqueta se convierte, durante su visualización, en un enlace tal que, cuando se pulsa, permite ver otras anotaciones que también contienen esa etiqueta.

El uso de micro-*blogging* proporciona a los profesores un mecanismo para obtener información diaria sobre los alumnos, sus intereses o sus dudas, y así conocer de una manera sencilla el ritmo de clase. La experiencia en [14] estudia el uso de *Twitter* para recolectar dudas acerca del curso académico. Explica la evolución de los experimentos con consideraciones sobre cómo organizar las discusiones en grupos, temas, o como enseñar a los estudiantes a usar esta herramienta de forma efectiva.

Sin embargo, en muchos de estos estudios se destacan algunas limitaciones o inconvenientes de este tipo de herramientas en el ámbito educativo.

- El más destacable es la necesidad de una persona que supervise la actividad de los usuarios. En este caso, el profesor deberá llevar un control de manera que pueda centrar y orientar las discusiones de sus alumnos, y hacer de la herramienta un buen recurso de aprendizaje [26].

- Por otro lado, el límite de caracteres provoca que en ocasiones, preguntas o ideas complejas no se puedan explicar con claridad [12, 13, 14].
- Otra "barrera" a destacar es la necesidad de una formación previa tanto para alumnos, como para profesores sobre la plataforma escogida. Esto contribuirá a un aprovechamiento mayor de la herramienta en relación con el objetivo que se persigue [32].
- Otro inconvenientes la frecuencia de aparición de errores tipográficos, provocados en la mayoría de los casos por acceder a la herramienta desde dispositivos móviles. Esto en ocasiones dificulta la comprensión de las ideas [26, 30].
- Un problema importante es la falta de privacidad del estudiante, ya que por lo general, las anotaciones son públicas [26].
- Otra limitación presente en algunas plataformas es la posibilidad de edición de las anotaciones por parte de los alumnos, con el consecuente riesgo que conlleva de cara a la evaluación por parte del profesor [32].
- Por último, destacar el inconveniente provocado por el uso de herramientas comerciales, como *Twitter*. La utilización de este tipo de sistemas dificulta su adaptación al ámbito educativo, ya que sus términos y condiciones, imponen restricciones sobre sus contenidos, que en ocasiones, pueden ir en contra de los métodos docentes [26].

Además de las iniciativas comentadas durante esta sección, a continuación se comentan algunos de los experimentos de este tipo más relevantes de acuerdo al objetivo de este proyecto.

En [20], se comenta el estudio realizado sobre 15 estudiantes de *Sheffield Hallam University*, en el que se propuso a los participantes utilizar *Twitter* de una manera diferente. En vez de contestar a la típica pregunta "*What are you doing?",*debían responder a la cuestión "*Where are you learning?".* A pesar de que los *tweets* de cada participante variaban en gran medida, todos ellos superaron las expectativas. Los autores consiguieron a partir de una actividad no tradicional, involucrar a los estudiantes en el desarrollo y la planificación de la universidad.

El siguiente experimento se realizó en *Marquette University (Wisconsin, EE.UU.)*[21]. En este caso, *Twitter* fue utilizado para compartir información entre la comunidad educativa. G*ee Ekachai*, profesor de la universidad, comenzó a utilizar *Twitter* para que los alumnos fueran elaborando anotaciones en vivo acerca de las clases. El estudio destaca resultados positivos en los que a través de este ejercicio, se ayudaba a los estudiantes a desarrollar habilidades fundamentales, como escuchar, recopilar información, e incluso hacer más de una tarea a la vez. No obstante, hubo más profesores como *McGee Young*, que no la consideraron útil, destacando como limitación, la dificultad que plantea el que un gran número de personas, pueda interactuar de manera conjunta con una aparición individual significativa por parte de todos los miembros.

En la *Universidad de Ciencias Aplicadas de Upper Austria*, se midió el grado en que este tipo de herramientas ayudan en el proceso de aprendizaje, tanto informal como orientado [22]. Se concluyó que el micro-*blogging* debía ser visto como una nueva forma de comunicación que soporta completamente el aprendizaje informal derivado de las clases.

Otras investigaciones como [24], un estudio en el que se han investigado las redes de micro-*blogging* utilizando *TAM* (Technology Acceptance Model), han validado el uso de este tipo de herramientas en combinación con las clases tradicionales. No obstante, han encontrado algunas limitaciones como el "ruido" que puede acabar alcanzándose en una red en la que se tratan temas muy diversos.

En [31], se estudia la interacción de los estudiantes con la red social de micro-*blogging* por excelencia, *Twitter*. En las evaluaciones estadísticas llevadas a cabo en el estudio, se confirma la tendencia de los alumnos a interactuar más entre estudiantes de niveles similares, y a prestar más atención a estudiantes con un mayor número de logros.

Entre los experimentos previos de este tipo, destacan por encima del resto el uso de este tipo de herramientas en el ámbito de *marketing*, y de periodismo, por ejemplo en la *Universidad de California-Berkeley y Stanford* [23].

Para complementar a los experimentos anteriores, y utilizando conclusiones obtenidas en algunos de estos trabajos, surge *BoloTweet*, que aquí se denomina 1.0 para distinguirlo de la contribución actual, que es *BoloTweet 2.0*. B*oloTweet 1.0* propone un sistema de uso basado en micro-*blogging* totalmente innovador, y diferente al ofrecido por cualquier iniciativa anterior de este tipo según la literatura revisada. En concreto, se propone que el profesor pueda evaluar *tweet* a *tweet* el trabajo del estudiante. De acuerdo a los resultados obtenidos en [31], la incorporación de un *feedback* que proporcione información a los estudiantes acerca del trabajo de otros compañeros, parece clave para lograr una buena herramienta de apoyo a la enseñanza que sirva como fuente de conocimiento para los alumnos. También se propone una distinción entre profesor y alumno, algo inexistente en las iniciativas revisadas, y que sin embargo sí ofrece *BoloTweet 1.0*.Seplanteaademás una metodología de trabajo muy concreta, algo que no se aprecia en el resto de trabajos.

Los primeros resultados experimentales de esta herramienta arrojaron resultados prometedores [25], lo que justifica continuar y evolucionar sus principios. La incorporación de nuevas funcionalidades, como las detalladas en esta memoria, así como la resolución de carencias observadas durante el uso previo del sistema, abren una nueva oportunidad de investigación para validar, o redefinir la metodología propuesta.

2.3 Tecnologías de soporte para micro-*blogging*

Twitter no es la única plataforma sobre la que se puede aplicar micro-*blogging.* Aparte de *Twitter*, actualmente las más relevantes son *StatusNet*, *Jisko, Sharetronix* y *JaikuEngine*.

Jisko, a pesar de sus buenos comienzos, a día de hoy es una alternativa abocada a la desaparición. Esta red social, además de tener una interfaz demasiado sencilla, y de no ofrecer toda la funcionalidad requerida por *BoloTweet*, es un proyecto discontinuado y cerrado hace ya casi 4 años. El cierre y la no evolución de esta herramienta, han hecho que esta opción sea descartada.

Por otro lado *JaikuEngine*, es un servicio de micro-*blogging* comprado por *Google* y desarrollado sobre la plataforma *App Engine*. Sin embargo, el fracaso de esta plataforma en relación con su principal competidor *Twitter*, provocó su liberación por parte de *Google*, y su cierre a finales de 2011. Esta herramienta actualmente no tiene ninguna comunidad de desarrollo relevante, ni se espera una evolución de la misma, por lo que se ha decidido no utilizarla.

ShareTronix ofrece un sistema de software social desarrollado en PHP, y de código abierto, con una comunidad cada vez mayor, y en constante evolución. S*hareTronix* plantea un sistema de desarrollo basado en *hooks*, muy similar al propuesto por *Status.Net*. Sin duda, es una herramienta con un enorme potencial, y que probablemente, dentro de un tiempo, sea una buena alternativa a *Status.Net*.

Sin embargo, actualmente *Status.Net*, anteriormente *Laconica*, es la herramienta de código abierto más afianzada, aún en evolución, y con la mayor comunidad de usuarios en sistemas de este tipo. Además, está desarrollada en PHP, al igual que la alternativa anterior, y utiliza tecnologías como MySQL, JavaScript, JQuery o CSS, muy acertadas para la implementación de *BoloTweet 2.0*.

Identi.ca, es la página web más importante desarrollada a partir de *Status.Net*. Sin embargo, hace apenas un año, migró al software *pump.io*, creado por el propio autor de *Status.Net*, pero desarrollado en *Node.js*, una tecnología poco dominada para poder afrontar este proyecto.

Además, otro de los motivos por los que se ha decidido apostar de nuevo por*Status.Net*, ha sido su interfaz y sus funcionalidades, ofreciendo una apariencia simple y fácil de manejar, y algunas características muy interesantes para un contexto educativo, como "*Eventos*", "Encuestas" o "*Preguntas*".

En cuanto a la versión, se ha decidido desarrollar sobre la última *release* (*1.1.1*). Mencionar que *StatusNet* está desarrollado bajo licencia *Creative Commons*, de manera que puede usarse libremente.

En la *Ilustración 3*, se muestra la licencia escogida para *BoloTweet*.

Reconocimiento – NoComercial – CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Ilustración 3 - Licencia BoloTweet

Esta licencia se aplica a todas las anotaciones y contenidos generados en *BoloTweet*, de manera que cualquiera puede utilizarlos siempre y cuando no sea con fines comerciales, y se mantenga reconocimiento al autor.

No obstante, como se puede ver en la *Ilustración 4*, en *BoloTweet 2.0*se indica la licencia escogida en el pie de página.

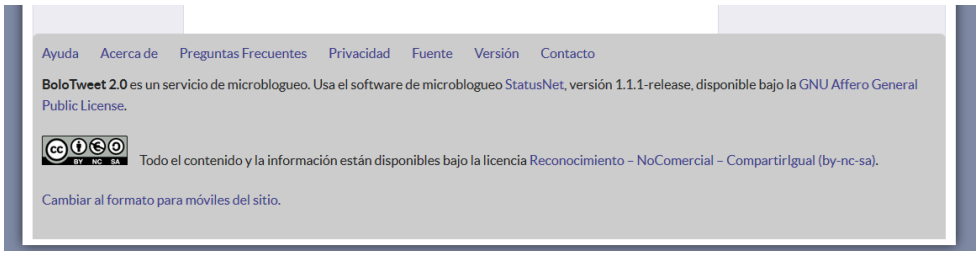


Ilustración 4 - Captura del pie de página de BoloTweet

2.4 Conclusiones

A día de hoy, el mayor aumento en las redes sociales se está produciendo en las redes de micro-*blogging*. *Twitter* es el sistema de este tipo con más relevancia en la actualidad. Por las características que ofrecen este tipo de redes, están siendo utilizadas en trabajos de soporte a la docencia. Sin embargo, la literatura hace un uso de los sistemas de micro-*blogging* muy restrictivo. Todos se limitan a usar la interfaz de *Twitter*, que no permite hacer cualquier cosa, como obtener conversaciones completas o valorar tweets.

BoloTweet 1.0 partió de una implementación distinta de un sistema de micro-*blogging* que le permitió definir mejor la forma de dar clase y ofrecer un soporte más preciso. Además, favorece la privacidad, ya que su uso no está sometido a los términos que impone una empresa.

Sobre *Bolotweet 1.0* se plantea un *BoloTweet 2.0* con mejoras sobre las deficiencias de *BoloTweet 1.0* ya citadas en la introducción, y con nuevas funcionalidades.

Entre la literatura revisada, el método docente aplicado en casi todos los experimentos es el mismo. Por un lado, el profesor hace de moderador de la clase, controlando el correcto uso de la herramienta por parte de sus alumnos. Por lo general, las anotaciones son redactadas durante el desarrollo de las clases, y en ningún caso se plantea ningún tipo de *feedback* hacia las mismas.

En cuanto a las plataformas existentes, destacar como las más importantes *StatusNet, Jisko, ShareTronix y JaikuEngine*. Entre todas ellas, se ha decidido continuar con *StatusNet*, por ser actualmente la que ofrece una estabilidad y una funcionalidad por encima del resto, y por adaptarse en gran medida al comportamiento requerido para el uso de este tipo de herramientas en el ámbito educativo.

3 Una propuesta de apoyo docente basado en microanotaciones

BoloTweet 2.0 se construye sobre la metodología de apoyo a la docencia propuesta por la primera versión del sistema, *BoloTweet 1.0*. La versión inicial concebía dentro de la clase un uso basado en resúmenes, críticas y dudas distinguiendo unos de otros mediante el uso de etiquetas o *hashtags*. Estos principios se reúsan adaptándolos a la nueva interfaz que propone la versión actual de *Status.Net* y que permite tratar estas necesidades de forma específica sin utilizar etiquetas.

Aparte, a esta metodología, que posteriormente se detallará, se le han añadido una serie de funcionalidades que solucionan algunas de las carencias encontradas en la versión anterior. Relacionadas con la metodología, se incorporan como novedad el concepto de Tareas, Apuntes, así como una reingeniería dela funcionalidad destinada a la calificación de anotaciones. Otros conceptos como las encuestas, las preguntas y los eventos son proporcionados por *Status.Net*.

La metodología de apoyo docente se expresa aquí como directrices o recomendaciones para profesores y alumnos. Se trata de una serie de herramientas de apoyo a la docencia que permiten hacer un seguimiento diario del trabajo de los estudiantes.

La forma de aplicar *Bolotweet 2.0* consiste en seguir estas directrices de uso respetando la filosofía de *BoloTweet*:

- Los estudiantes podrán saber qué valor aporta el profesor a las anotaciones. Un estudiante debe saber pronto si lo que cree aprender en una clase coincide con lo que el profesor espera que aprenda. Otros estudiantes pueden usar esta valoración también para valorar sus propias impresiones de cómo es la clase.
- Cualquier trabajo planteado en *BoloTweet* debe poder ser realizado dentro del tiempo de clase. Esto se aplica al trabajo de profesor y del estudiante.
- En general, el método docente consistirá en apoyar el desarrollo de la actividad de clase mediante herramientas concretas basadas en el micro-*blogging*. El método más aplicado en la experimentación realizada consiste en pedir a los estudiantes que resuman alguna de las ideas vistas en clase dentro de las limitaciones impuestas por los *tweets*.

A continuación se comentarán una serie de directrices generales, aplicables para cualquier usuario de *BoloTweet 2.0*, así como unas pautas orientadas por un lado a alumnos, y por otro a profesores.

3.1 Directrices generales

Para orientar el uso de este tipo de herramientas a la docencia, es necesario imponer unas ciertas normas que definan el comportamiento de cada elemento de la red social en un contexto educativo:

- Por ser alumnos de la UCM, debe respetarse el código de conducta de la Universidad Complutense de Madrid.
- Cada asignatura está representada en *BoloTweet* por un grupo único, al que pertenecerán todos los alumnos y profesores de la asignatura.

- Este grupo tiene un *timeline*, o panel principal, donde se van agrupando las anotaciones enviadas por los alumnos.
- Cada grupo tiene un *nickname* asignado que le identifica, y que permite a los estudiantes referenciarle de manera única. Por ello, es importante que el nombre tenga relación con la asignatura, ya que de lo contrario, podría producir confusiones.
- Toda anotación enviada a un grupo, pasa a formar parte del contenido del mismo, y se hace visible para cualquier persona que acceda a la sección dedicada a tal grupo **.
- Únicamente pueden enviar *tweets* a una asignatura, y considerarse como tal, los miembros propios del grupo.
- Cualquier anotación con referencia a un grupo, puede ser calificada por el profesor, o los profesores pertenecientes a tal grupo.
- Para enviar un mensaje a una determinada asignatura, *BoloTweet* ofrece dos opciones:
  - A través de la propia anotación, utilizando el carácter '!', seguido del *nickname* del grupo.
  - Utilizando el selector de grupo asociado al campo de anotaciones (ver *Ilustración 5*).

** *Una excepción son los grupos privados, cuyo contenido únicamente es visible por sus miembros.*

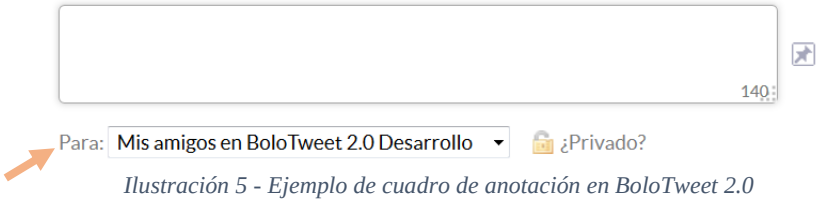


Ilustración 5 - Ejemplo de cuadro de anotación en BoloTweet 2.0

3.2 Directrices para alumnos

Además de las directrices previamente comentadas, se propone una metodología de trabajo asociada al alumno, que permita utilizar *BoloTwet 2.0* con un cierto orden, y con un comportamiento común entre todas las asignaturas.

Como pautas generales, se proponen el *resumen diario*, la *crítica a la lección*, y las *preguntas*. Como complemento a estas, se explican algunos de los mecanismos incorporados en *BoloTweet 2.0* para facilitar su uso.

La explicación en detalle de cada función, y herramienta propuesta, está explicada en el *Manual de Uso - Usuario*.

3.2.1 Resumen diario

En cada clase, se propone que, durante los últimos 10 minutos, los alumnos realicen una anotación con la idea que consideren más relevante acerca de lo expuesto en clase con la limitación de los 140 caracteres y respetando la buena ortografía y gramática. Para ello, es necesario que reflexionen acerca del contenido recibido, y lo sinteticen de la mejor forma posible.

Para realizar esto con una estructura común, y para poder distinguir anotaciones por días y asignaturas, se propone que:

- Cada anotación de este tipo debe ir dirigida al grupo que represente a la asignatura en cuestión, utilizando uno de los dos métodos indicados anteriormente en la sección 3.1.

- Deberá además contener una etiqueta (*hashtag*) con la fecha del día (***#aaaa-mm-dd***).
- Se podrá añadir cualquier otra etiqueta adicional que mejore el entendimiento del *tweet*, y que además sirva para relacionarlo con otros similares.

En la *Ilustración 6* se puede ver un ejemplo de resumen diario con el formato y la metodología propuesta.

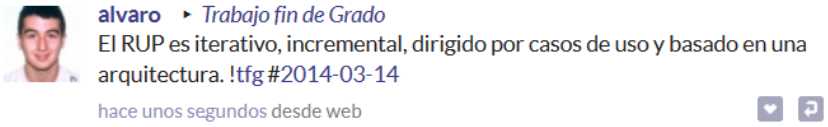


Ilustración 6 - Ejemplo resumen diario

La anotación directa del resumen no está sujeta a ningún control. Más adelante se propondrá el mecanismo de tareas como soporte a la elaboración de resúmenes. El resumen diario es la herramienta principal de seguimiento del progreso del estudiante. Combinado con la evaluación mediante puntos, se consigue discretizar los avances y localizar de forma temprana estudiantes que no están adquiriendo los conceptos que se esperaba.

3.2.2 Tareas

La realización de los resúmenes diarios está sujeta a errores. Por ejemplo, que se confundan en la redacción de la fecha usando un formato ligeramente distinto, o que no se indique correctamente el grupo al que va dirigido. Para evitar esto, se plantea el siguiente mecanismo basado en tareas.

Las tareas las inicia el profesor y se proponen como una manera automatizada de generar resúmenes diarios, u otro tipo de actividad, ya que automáticamente cada anotación incluye:

- Etiqueta con la fecha precargada con el formato esperado.
- Grupo de destino seleccionado (No se puede modificar).
- Etiqueta opcional, en caso de que el profesor la haya indicado en la tarea.

El mecanismo de uso de las tareas en alumnos es el siguiente:

- El profesor crea una tarea para hacer un resumen, por ejemplo, de lo expuesto en clase.
- Cuando un alumno tenga una tarea pendiente, automáticamente recibirá un aviso en el menú principal (*Ilustración 7*).
- Al acceder al menú de Tareas, se mostrará el listado de las tareas asociadas al usuario, y aún no completadas (*Ilustración 8*).
- Cada tarea puede ser rechazada, eliminando por tanto su vínculo con el usuario, o bien completada, generando el resumen diario asociado a una clase.
- En caso de completar la tarea, únicamente deberá anotar la idea en el cuadro *precargado*, ya que el resto de directrices se completan automáticamente (*Ilustración 9*).

- Una vez rellenada, la anotación aparecerá en el *timeline* del grupo como una anotación convencional.

Ilustración 7 - Aviso de tareas pendientes para el alumno

Ilustración 8 - Ejemplo de listado de tareas pendientes

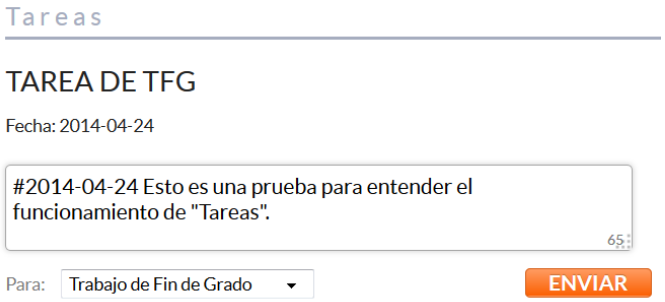


Ilustración 9 - Ejemplo de Tarea

Otra de las opciones propuestas consiste en la elaboración de anotaciones de crítica hacia una asignatura, o clase concreta. El proceso de creación es similar al resumen diario.

- Cada anotación de este tipo debe hacer referencia al grupo en cuestión, utilizando alguno de los mecanismos indicados anteriormente en la sección *3.1*.
- Deberá además contener un *tag* con la fecha (***#aaaa-mm-dd***).
- Para que la anotación sea considerada como una crítica, es necesario incluir la etiqueta "***#crit***".
- Cualquier otra etiqueta que ayude a referenciar el problema, o a relacionarlo con anotaciones similares, se puede incluir en forma de *tag*.

En la *Ilustración 10*, se puede ver un ejemplo completo de crítica a una determinada asignatura, en este caso al grupo "Trabajo de fin de Grado" (tfg), y el día 10 de Marzo de 2014.

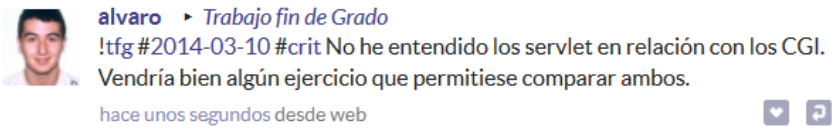


Ilustración 10 - Ejemplo de crítica a la lección

Animar a los estudiantes a evaluar de forma crítica es parte de la vida universitaria. La crítica es siempre necesaria, y puede ser origen de interesantes debates. Además, la naturaleza del micro-*blogging* permite la réplica y contrarréplica. Aunque la línea argumental es difícil de seguir, puede ser una herramienta alternativa a los foros de discusión en este sentido.

3.2.3 Apuntes

Como novedad en *BoloTweet 2.0*, se proporciona un nuevo mecanismo basado en Apuntes. La intención es proporcionar al alumno la opción de recopilar las anotaciones propias, o de los compañeros. Los apuntes se generan por asignaturas, de manera que no se puedan mezclar ideas y comentarios de grupos distintos.

Para la utilización de la herramienta, se proponen dos opciones:

- Apuntes automáticos. Estos apuntes contendrán las anotaciones mejor valoradas por parte de los profesores vinculadas a un determinado grupo.
- Apuntes personalizados. Este tipo de apuntes permite personalizar los apuntes, permitiendo al alumno escoger algunos parámetros tales como el autor de las anotaciones, las etiquetas, o las puntuaciones (*Ilustración 11*).

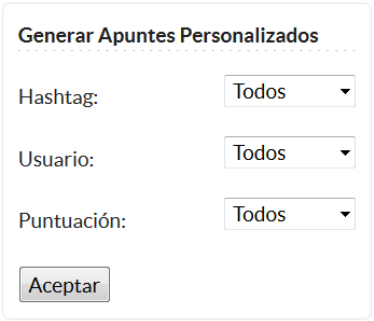


Ilustración 11 - Formulario de personalización de apuntes

Los apuntes son un material de estudio, pero en ningún caso sustituyen al proporcionado por el profesor.

Los apuntes son una referencia para los estudiantes y los profesores que les permite tener idea del transcurso de la clase. A los estudiantes, les resume lo que otros compañeros producen utilizando *BoloTweet*. Puede aportarles vistas desde otros ángulos que no han sabido entender desde las explicaciones del profesor, y puede servir como herramienta para agrupar y recopilar las propias ideas para obtener un material de estudio complementario.

3.2.4 Preguntas

En caso de querer indicar una pregunta, y que esta sea reconocida como tal, *Status.Net* ofrece un mecanismo personalizado a través del cual se consigue un comportamiento propio de una pregunta, con posibilidad incluso de elección de respuestas (*ver Ilustración 12*).En este caso, se proponen las siguientes indicaciones:

- Es necesario seleccionar el grupo al que va dirigida utilizando alguno de los métodos explicados en la sección *3.1*.
- Debe especificarse un título apropiado que identifique la pregunta, no se admiten generalidades.
- El cuerpo de la pregunta deberá indicarse de manera breve y concisa, ofreciendo toda la información que pueda ser necesaria para su resolución.
- En caso de querer indicar una fecha, es posible hacerlo a través de un *tag* con el formato "***#aaaa-mm-dd***".
- Una vez se haya resuelto la pregunta, es responsabilidad del autor marcar como "Mejor" la respuesta escogida, de manera que pueda servir de referencia para el resto de compañeros.

Ilustración 12 - Ejemplo de pregunta y respuesta

La aparición de preguntas relacionadas y derivadas de las clases es un aspecto muy común en el ámbito educativo. Por ello, se considera de gran utilidad el soporte de un mecanismo dedicado a preguntas dentro de un sistema de micro-*blogging*. A partir de esta herramienta, es posible generar anotaciones con carácter de preguntas, sin necesidad de utilizar ningún tipo de etiqueta específica. Además, el sistema de elección de respuestas permite al resto de usuarios obtener conocimiento a partir de las preguntas elaboradas por otros compañeros.

3.2.5 Puntuaciones

Se mantiene en *BoloTweet 2.0* el concepto de *feedback* basado en puntuaciones. Las calificaciones van vinculadas a cada anotación, y pueden ser vistas por cualquier usuario de *BoloTweet*. La puntuación se

muestra en un recuadro individual asociado al *tweet*, en el que se muestra la puntuación y el nombre del profesor que la ha realizado.

En caso de que una anotación haya sido puntuada por más de un profesor, es posible ver el listado detallado de puntuaciones pulsando en el recuadro con la anotación (*Ilustración 13*).

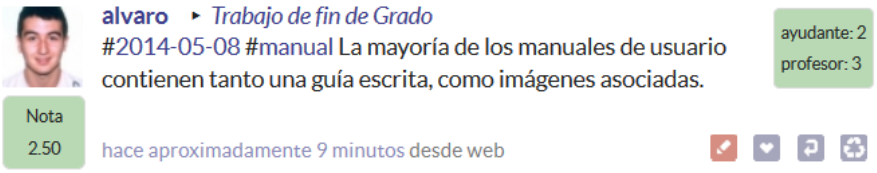


Ilustración 13 - Ejemplo de anotación puntuada y de desglose de puntuaciones

Uno de los distintivos de *BoloTweet*, desde su versión *1.0*, consiste en proporcionarle al alumno información valiosa acerca de su actividad en *BoloTweet*. Esto se consigue a través de puntuaciones, que sirven de *feedback* tanto para él como para el resto de compañeros. El objetivo de esto radica fundamentalmente en que el alumno sepa si ha entendido bien o no un concepto. De esta manera, se motiva al alumno para que asista a resolver dudas con el profesor derivadas de puntuaciones negativas.

3.2.6 Listados

Aprovechando las puntuaciones asociadas a cada anotación, *BoloTweet* ofrece un sistema de listados de calificaciones para cada asignatura (*Ilustración 14*).

- El listado de una asignatura únicamente es visible para sus miembros. De esta manera se consigue evitar la falta de privacidad existente en *BoloTweet 1.0*.
- Los listados muestran los alumnos del grupo ordenados por puntuación total mayor. La puntuación se crea a partir de las valoraciones recibidas por parte de los profesores en las anotaciones de un alumno en un determinado grupo.
- El *ranking* de estudiantes de cada grupo no muestra la puntuación total asociada a cada alumno.

Ilustración 14 - Ejemplo de listado de calificaciones

El objetivo del listado de puntuaciones en relación con los alumnos es que puedan ser utilizados para conocer su ritmo en la asignatura, y en consecuencia esforzarse e implicarse más. Como objetivo secundario, a través del listado los alumnos pueden conocer a qué compañeros podrían acudir para resolver alguna duda. Además, si se accede al nombre de cualquier alumno desde el listado, se muestra una página con todos los *tweets* puntuados de ese estudiante en el grupo, lo que puede servir de mucha ayuda para conocer y obtener conocimiento a partir de las anotaciones enviadas por los compañeros mejor calificados.

3.3 Directrices para Profesores

Mientras que los estudiantes proporcionan el conocimiento del sistema, las anotaciones, el profesor dirige la creación de dicho conocimiento y lo criba mediante una valoración numérica del mismo.

3.3.1 Tareas

Se incorpora en *BoloTweet 2.0* el concepto de Tareas. Además de automatizar la generación de resúmenes diarios por parte de los alumnos, se proporciona un mecanismo a través del cual realizar un seguimiento del trabajo continuo de los alumnos. El uso principal realizado en los primeros ensayos ha estado orientado al soporte de los resúmenes diarios vinculados a cada clase, pero su uso puede ampliarse a otras áreas.

A continuación se detallan una serie de indicaciones en cuanto a la creación de tareas:

- El mecanismo de creación de tareas, así como su cancelación, modificación o reapertura están explicados con detalle en el *Manual de Uso - Usuario*.
- Cada tarea está vinculada a una clase, por lo que solamente es posible crear una tarea por día y grupo.
- De manera opcional, se podrá indicar una etiqueta relacionada con la tarea, y que lógicamente, tenga relación con la materia impartida en una determinada clase. Esta etiqueta se incluirá en todos las tareas completadas por los alumnos. Esta *tag* en un futuro se puede utilizar para señalar usos especiales.
- Las tareas iniciadas, así como el número de *tweets* recibidos para cada una de ellas, puede ser revisado a través del histórico de tareas asociado a cada asignatura (*Ilustración 15*).

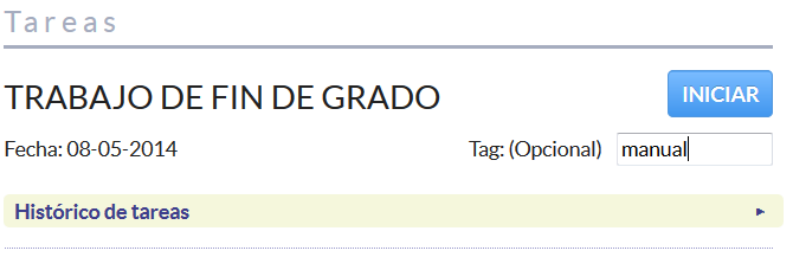


Ilustración 15 - Ejemplo de creación de tareas

La herramienta de tareas es algo que se incorporó tarde al proyecto. Se ha usado principalmente como remedio de los defectos del sistema que se venía usando para generar resúmenes en *Bolotweet 1.0*. Sin embargo, admite otros usos. Cada uso puede ser caracterizado por una etiqueta pre-convenida entre profesores y estudiantes y que puede incorporarse como el *hashtag* opcional que permite la tarea. Por ejemplo, si se desea una crítica de todos los estudiantes, para valorar su capacidad de evaluación de los contenidos de clase, se puede meter el *hashtag* ***crit*** y seguir los principios de crítica de la sección *0*.

3.3.2 Puntuaciones

Una de las funcionalidades únicas que ofrece *BoloTweet* desde su versión 1.0, es la posibilidad de ofrecer un *feedback* personalizado a cada anotación, de una manera rápida y efectiva.

- Las puntuaciones existentes en BoloTweet varían desde 0 a 3 puntos, siendo 0 la peor nota, y 3 la mejor.
- Un profesor únicamente puede puntuar las anotaciones dirigidas a los grupos de los cuales es profesor.
- El procedimiento de calificación, explicado en el *Manual de Uso - Profesor* se realiza a través de botones individuales para cada una de las 4 posibles notas.
- Es importante que se aprovechen los 10 minutos de cada clase en los que los alumnos generan el resumen diario, para evaluar las calificaciones de días anteriores.
- Cualquier puntuación puede ser modificada por el propio autor siempre que se necesite (*Manual de Uso - Profesor).*
- Se ha incorporado en *BoloTweet 2.0 l*a multiplicidad de profesores. De esta manera, si en una asignatura existe más de un profesor, cualquier de ellos, o incluso todos, pueden puntuar una misma anotación.

En la *Ilustración 16* se pueden observar los botones de calificación asociados a cada *tweet*.

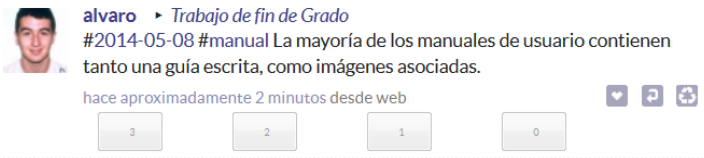


Ilustración 16 - Ejemplo de botones de calificación

La forma en que se valoran los *tweets* depende del tiempo del que disponga el profesor. Puede optar por puntuar directamente o bien, cuando la puntuación sea baja, responder a la anotación para indicar qué fue mal. En cualquier caso es importante convenir en que algo que está puntuado con la máxima nota es algo que hubiera podido decir el mismo profesor. Por ello, un tweet valorado así se convierte en referencia de confianza para el resto de los estudiantes. El trabajo de puntuar *tweets* lo pueden hacer varios profesores. De hecho, como se ha comentado anteriormente, un *tweet* puede recibir más de una calificación. En este caso, se calcula una media aritmética ponderada donde tiene mayor peso a la puntuación dada por el profesor responsable de la asignatura.

3.3.3 Listados

Utilizando las valoraciones asociadas a cada *tweet*, se ha generado un sistema de listado de calificaciones, a través del cual es posible visualizar para cada asignatura, un ranking de alumnos ordenado por mayor puntuación total.

A través de este listado (*Ilustración 17*), el profesor puede conocer de manera instantánea la actividad de un alumno en un determinado grupo.

Además, en *BoloTweet 2.0*, el profesor puede exportar el propio listado a un fichero CSV, para posteriormente tratar los datos o incluirlos en cualquier otra herramienta (*Ilustración 17*).

Ilustración 17 - Ejemplo de listado de puntuaciones

Estos listados sirven para, de forma abreviada, dar una idea del progreso de la clase. Una clase que estudie y haga el trabajo de forma regular y constante debería mostrar una lista de puntuaciones donde todos los estudiantes están cerca unos de otros. Los estudiantes que tengan dificultades en la clase deberían destacar del resto con puntuaciones alejadas de la media.

Además, de la misma forma que se ha explicado en la sección *3.2.6*, si se accede al nombre de cualquier alumno a partir de un listado, se muestra una página con el conjunto de todas las anotaciones puntuadas de ese estudiante en el grupo que corresponda. Los profesores observarán además una sección con estadísticas relativas al alumno (*Ilustración 18*).

Ilustración 18 - Ejemplo de tweets puntuados para un alumno junto con estadísticas

Con esta funcionalidad es posible revisar de una manera sencilla la actividad de un determinado alumno en un grupo, llevando así una evolución continúa del estudiante si se desea.

3.3.4 Encuestas

Status.Net ofrece un mecanismo de encuestas personalizado (*Ilustración* 19), a través del cual se pueden lanzar cuestiones y recibir información acerca de cada respuesta. Si el profesor así lo considera, podrá iniciar una nueva encuesta dirigida a sus alumnos, utilizando el proceso de creación detallado en el *Manual de Uso - Profesor*. Es importante que se detalle bien la pregunta, las opciones, y que se seleccione el grupo para el cual va dirigida utilizando el selector explicado en la sección *3.1*.

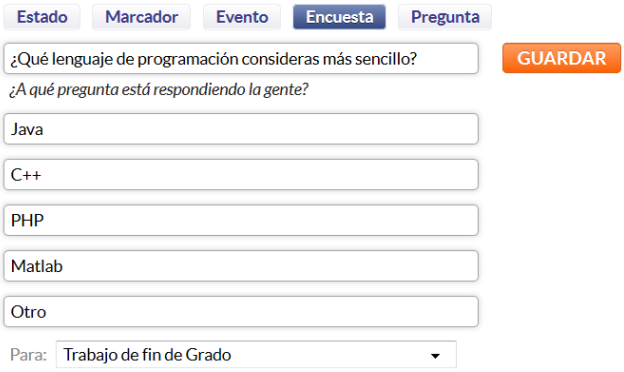


Ilustración 19 - Ejemplo de Encuesta

El resultado de las encuestas es compartido por todos los participantes. La utilidad de las encuestas es algo que no se ha podido constatar en los ensayos realizados. No obstante, hay profesores que la demandan, por lo que se ha dejado disponible.

3.3.5 Eventos

Por último, *Status.Net* ofrece un mecanismo de creación de Eventos (Ilustración 20) a través del cual poder recibir información acerca de la asistencia individualizada de cualquier usuario. El mecanismo de creación de eventos, detallado en el *Manual de Uso - Profesor*, está restringido únicamente a profesores. Es importante que se rellenen todos los campos de forma más precisa posible, y que se dirija al grupo en cuestión, a través del selector de grupo explicado en la sección *3.1*.

Ilustración 20 - Ejemplo de Evento

Por cuestión de tiempo, los eventos no han sido utilizados, por lo que no se ha podido comprobar su utilidad. No obstante, su comportamiento se ajusta perfectamente a eventos educativos, tales como charlas, conferencias o incluso para recoger comentarios o discusiones sobre la mismas, por lo que se ha visto conveniente mantenerlo para ser validado en futuros usos de *BoloTweet*.

3.4 Conclusiones

Para utilizar de forma correcta un sistema de micro-*blogging* en el ámbito educativo, es necesario imponer un conjunto de directrices que evite en gran medida algunas de las limitaciones e inconvenientes provocados por el uso de este tipo de redes sociales y que han sido comentados en la sección*2*.

Por un lado, es importante relacionar cada asignatura con un grupo propio. Cada grupo deberá tener un nombre único y relacionado con el tema de la asignatura. A este grupo, pertenecerán todos los estudiantes y profesores vinculados a la asignatura.

Los alumnos deben seguir una serie de indicaciones básicas que permitan utilizar el sistema de una manera homogénea entre todos los usuarios y asignaturas. Todas ellas han sido comentadas en la sección 3.2. En cuanto al método de ayuda a la docencia, se proponen resúmenes diarios, críticas a la lección y preguntas. Se proporcionan nuevas herramientas como los Apuntes o las Tareas.

Los profesores de la misma manera, deben seguir unas pautas propuestas. A través del mecanismo de puntuaciones ofrecido, los profesores deben proporcionar un *feedback* a las anotaciones de sus alumnos. Se aconseja utilizar los 10 minutos del resumen diario. Se proporcionan además herramientas de seguimiento como las Tareas o los Listados de puntuaciones, y se proponen el uso de mecanismos como Eventos o Encuestas.

Cada herramienta presentada aquí tiene el cometido de apoyar las actividades de clase, por ejemplo, reforzar los contenidos haciendo que los estudiantes resuman las ideas que ellos recuerdan. Durante la presentación de cada herramienta, se han dado unas breves indicaciones de cómo se ha venido usando. Algunas herramientas vienen por defecto con *Status.Net*. Estas son las encuestas, preguntas y eventos. Estas han sido integradas en el método de apoyo, pero su ensayo ha sido limitado. Más extenso ha sido el uso de las contribuciones originales de este trabajo, que son la generación automática de apuntes y las tareas. También han sido importantes la refactorización hecha del mecanismo de puntuación de *tweets* y de visualización de notas, creando profesores con un contexto asociado, y ofreciendo soporte para múltiples profesores y calificaciones.

4 Desarrollo de *BoloTweet2.0*

La funcionalidad a desarrollar se describe en forma de casos de uso. Cada caso de uso se explica brevemente con alusiones al método de apoyo docente que se quiere proveer y que fue presentado con anterioridad.

Cada caso de uso se resuelve con un diseño de un *plugin* para la plataforma *Status.net*. Entender su desarrollo implica comprender el funcionamiento de *plugins* propuesto por esta plataforma. Tras explicar los casos de uso, se explica por tanto de manera breve cómo funciona *Status.Net* y sus *plugin*. Después se diseñan las extensiones perseguidas para este proyecto y se completa la sección con una discusión sobre su implementación y despliegue.

La mayoría de la información relacionada con la implementación se encuentra disponible en la memoria de trabajo de la beca de colaboración con la que se ha soportado este desarrollo. Esta memoria está disponible en el *Diario Beca de Colaboración*.

La funcionalidad a desarrollar parte de trabajo previo en *Bolotweet 1.0*, por lo que también fue necesario migrar la funcionalidad implementada en el primer uso de *BoloTweet*, toda ella recogida en el plugin *Grades*. Ya que *Status.Net 1.1.1* propone una estructura definida para el desarrollo de plugins, ha sido necesario adaptar la versión anterior de *Grades* a esta nueva estructura aislada. Para ello, ha sido necesario desacoplarlo del *core* de *StatusNet*, ya que a pesar de utilizar ficheros separados, había partes del plugin que estaban ubicadas junto a otros ficheros del *core*, como reglas *css*, o scripts *JavaScript*. Todo este proceso se ha explicado con detalle en la tarea del *Diario Beca de Colaboración, Adaptación de grades a StatusNet 1.1.1*.

Aparte, se han descubierto y resuelto fallos internos de *Status.Net* y se han incorporado mejoras. Estos arreglos no son la principal contribución de este trabajo, pero son parte del esfuerzo necesario para lograr el funcionamiento actual de *BoloTweet 2.0*. Los detalles precisos están en el *Diario Beca de Colaboración*, sección *Tareas*. En esta sección se podrá encontrar una enumeración de los más relevantes.

4.1 Casos de uso para Bolotweet 2.0

Sobre esta plataforma, y mostrando las funcionalidades añadidas por los *plugins* incorporados, se definen casos de uso para los tres tipos de actores: profesores, estudiantes y administradores del sistema. Los casos de uso se listan y detallan a continuación.

Ver Ilustración 21

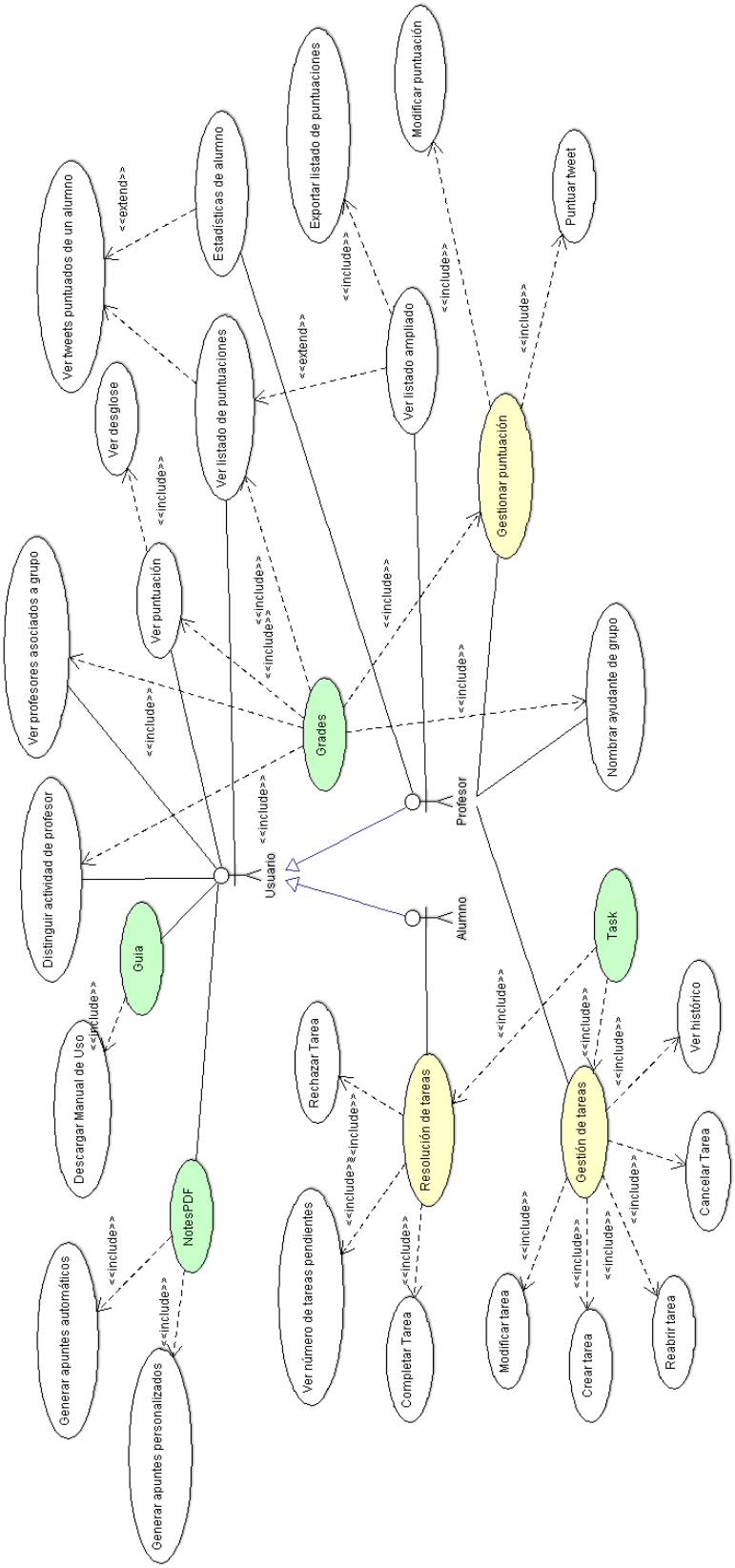


Ilustración 21. Casos de uso donde el actor principal es un Alumno o un Profesor. El actor Usuario representa un actor genérico que puede ser un alumno o un profesor.

Descargar Manual de uso: Para poder aplicar la metodología de apoyo docente propuesta en la sección 3, además de la posible información comunicada por parte de los profesores, ha sido necesario proporcionar un *Manual de Uso*, en el que se detallan las funcionalidades básicas del sistema, así como la metodología de ayuda propuesta. Para ello, se ha desarrollado el plugin *Guia*, a través del cual se ha implementado un mecanismo de descarga, que permite a cualquier usuario obtener el manual a través del formulario mostrado en la *Ilustración 22*.

La implementación de este plugin se puede revisar en la sección *Plugins* del *Diario Beca de Colaboración*.

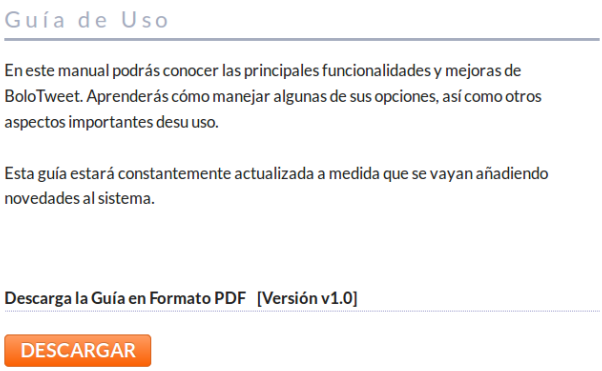


Ilustración 22 - Formulario de descarga de Manual de Uso

Generar apuntes automáticos. Una de las herramientas propuestas en la sección *3.2.3* hace referencia a la generación de apuntes desde los *tweets* de los estudiantes como forma de recopilar lo más relevante de acuerdo a las puntuaciones asociadas. Este nuevo concepto, incorporado por primera vez en *BoloTweet 2.0*, resuelve una de las carencias encontradas en el uso del sistema anterior, y es la falta de aprovechamiento de las ideas creadas por los propios usuarios. Para solventar esto, se ha desarrollado *NotesPDF*.

Los apuntes se generan por asignaturas, de manera que no se puedan mezclar *tweets* de diferentes materias, lo que podría llegar a provocar confusiones. En los apuntes automáticos, el sistema genera un fichero PDF a partir de las ideas mejor puntuadas, es decir, las de 2 y 3 puntos. El fichero resultante contiene un listado con todas esas anotaciones, además de una sección "autores" en la que se relaciona cada anotación con su autor.

Generar apuntes personalizados. En relación con el caso de uso anterior, otra de las opciones implementadas en el plugin *NotesPDF*, y comentadas en la sección *3.2.3*es la posibilidad de generar apuntes personalizados. Esto le proporciona al alumno la posibilidad de poder componer sus propios apuntes de la forma que mejor se adapte a sus necesidades. La personalización que se ofrece se basa en la elección del autor, de la etiqueta, y de la puntuación de las anotaciones a filtrar (*Ilustración 23*).

Si se ha seguido la metodología propuesta en la sección *3.2*, todas las anotaciones deben llevar un grupo asociado, una etiqueta con la fecha, y opcionalmente uno o más *tags* adicionales. De esta manera, es posible generar apuntes por día, o sobre cualquier tema que se requiera, utilizando el *tag* apropiado. Además, al poder seleccionar el autor, es posible generar únicamente apuntes propios. Estos ficheros PDF se generan y se descargan vía *OTA*, pudiendo generar dinámicamente la combinación que se desee.

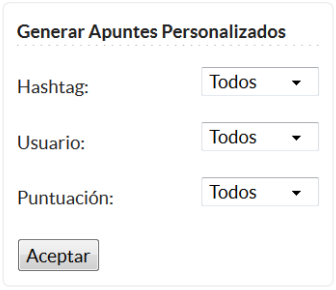


Ilustración 23 - Selección de apuntes personalizados

Ver número de tareas pendientes. Con la incorporación del mecanismo de Tareas, explicado en la sección *3.2.2*, los profesores ahora proponen resúmenes diarios a través de la creación de tareas. Para que el alumno pueda conocer de manera sencilla la existencia de éstas, sin necesidad de entrar expresamente al apartado correspondiente, se ha implementado un mecanismo de aviso a través del cual, el alumno puede visualizar en todo momento el número de tareas pendientes únicamente mirando la barra lateral izquierda. En la *Ilustración 24*, se puede ver un ejemplo de aviso de tareas.

Ilustración 24 - Ejemplo de aviso de tareas pendientes

Completar tarea. Como se detalla en la sección *3.2.2*, como acción sobre una tarea pendiente, existe la opción de completarla. Para ello, se deberá acceder al apartado de tareas, siempre y cuando tenga alguna pendiente, y pulsar el botón Completar. Automáticamente aparecerá un cuadro de anotación *precargado* con el tag con la fecha, el grupo al cual va dirigido, y opcionalmente, el *tag* indicado por el profesor. El alumno deberá sintetizar la idea correspondiente a la clase en cuestión, y escribirla en el cuadro de anotación. Tras pulsar enviar, la tarea se completará, y el resumen diario quedará enviado y sujeto a calificación (*Ilustración 25*).

Con esto se consigue automatizar el proceso de generación de resúmenes diarios, evitando fallos en la tipografía de las etiquetas requeridas, y homogeneizando la metodología de trabajo propuesta.

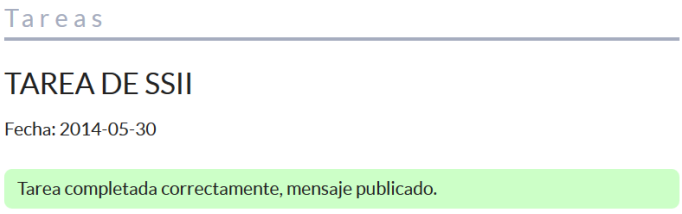


Ilustración 25 - Ejemplo de tarea completada

Rechazar tarea. En relación con los dos casos de uso anteriores, y tal y como se detalla en la sección *3.2.2*, se ofrece al alumno la posibilidad de rechazar cualquier tarea pendiente. Para ello, deberá pulsar el botón Rechazar asociado a la tarea, y a continuación confirmar el rechazo (*Ilustración 26*). De esta manera, la tarea se eliminará. Esto no es aconsejable, pero en ocasiones puede ser necesario. Por ejemplo, si el alumno ha enviado el resumen a través del método tradicional, es posible que quiera eliminar la tarea asociada a ese día concreto. No obstante, la tarea se queda vinculada al alumno internamente, pudiendo ver en cualquier momento las tareas rechazadas por un alumno concreto.

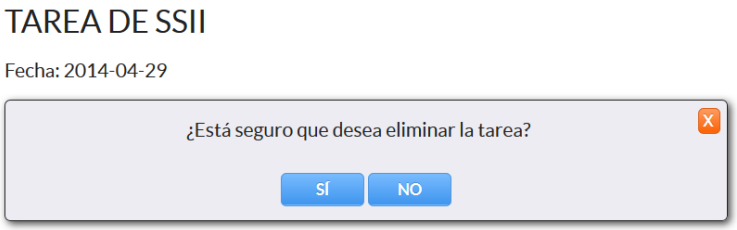


Ilustración 26 - Ejemplo de confirmación de rechazo de tarea

Crear tarea. Al incluir el mecanismo de tareas en *BoloTweet 2.0*, es el profesor quien debe iniciarlas en cada clase que lo requiera, para generar tareas pendientes en sus alumnos. Esta tarea sustituye al resumen diario, tal y como se explica en la sección *3.3.1*, y permite además llevar un seguimiento del número de resúmenes recibido en cada clase. Para crear una tarea, el profesor únicamente debe indicar un *tag* opcional vinculado con la materia del día, y pulsar el botón "Iniciar". A partir de ese momento, se creará una tarea asociada a los alumnos, con las características definidas por el profesor. Las tareas se crean por clase, lo que quiere decir que no es posible crear más de una tarea por grupo y día.

Ver histórico. Como se ha comentado en la sección *3.3.1*, cuando un profesor accede a la sección de tareas, se muestra un listado con las asignaturas vinculadas al profesor. Para cada una de ellas, se ha implementado un histórico de tareas (*Ilustración 27*). Este histórico genera un listado con las tareas creadas para una determinada asignatura, y a través de él es posible realizar acciones sobre ellas, como cancelarlas, reabrirlas, o modificarlas. Además, cada registro del histórico incluye información relevante sobre la tarea, como la fecha de creación, su estado, el *tag* opcional en caso de haberse incluido, o el número de alumnos que la han completado.

Ilustración 27 - Ejemplo de tareas en el perfil de profesor

Cancelar tarea. Si en algún momento el profesor desea cancelar una tarea enviada, tal y como se menciona en la sección *3.3.1* o como parte del proceso de modificación del *tag* asociado, es necesario utilizar el mecanismo de cancelación de tareas detallado en el *Manual de Uso - Profesor*. Únicamente es posible cancelar tareas que no hayan sido completadas aún por ningún alumno. Esto se controla automáticamente en el histórico de tareas de cada asignatura, ya que únicamente se muestra la opción de cancelar en las tareas con ninguna anotación recibida.

Reabrir tarea. Como se detalla en la sección *3.3.1*, el profesor puede reabrir cualquier tarea previamente cancelada. El proceso de reapertura está explicado con detalle en el *Manual de Uso - Profesor*. En este caso, una tarea cancelada se puede reabrir siempre que se desee, sea o no la fecha de creación de la misma. No obstante, la fecha asociada a la tarea será la original de creación, no la de reapertura.

Modificar tarea. Relacionado con los tres casos de uso anteriores, y como se menciona en la sección *3.3.1*, es posible que el profesor necesite en algún momento modificar alguna tarea previamente iniciada, para editar por ejemplo, la etiqueta opcional vinculada a la misma. De ser así, deberá utilizar el mecanismo de modificación detallado en el *Manual de Uso - Profesor*. Para modificar una tarea, es necesario realizarlo el día de creación de la misma, y que ningún alumno la haya completado aún, ya que de lo contrario, habría *anotaciones* asociadas a una misma tarea con diferente formato.

Puntuar tweet. De acuerdo a la metodología de apoyo docente propuesta en la sección *3.3.2*, una de las funcionalidades ofrecidas y características de BoloTweet es la posibilidad de ofrecer un *feedback* a cada anotación en forma de puntuación. El proceso de puntuación se detalla en el *Manual de Uso - Profesor*. Es importante mencionar que un profesor únicamente puede calificar anotaciones dirigidas a los grupos de los cuales es profesor (*contexto de profesores*). Las puntuaciones emitidas varían entre el 0, puntuación mínima, hasta el 3, puntuación máxima.

Relacionado con esto, en *BoloTweet 2.0* se ha implementado el soporte de múltiples profesores. De esta manera, una asignatura puede tener más de un profesor, y por tanto, una anotación puede ser puntuada por todos y cada uno de ellos.

Modificar puntuación. En el caso de emitir una puntuación incorrecta, y tal como se comenta en la sección *3.3.2*, se ha implementado un sistema edición de calificación. De esta manera, en cada anotación puntuada, el profesor dispondrá de un botón único para editar la nota emitida (*Ilustración 28*). Es importante conocer que la puntuación asociada a un *tweet* únicamente puede ser modificada por el autor de la propia calificación. El proceso de modificación de puntuación está detallado en el *Manual de Uso - Profesor*.

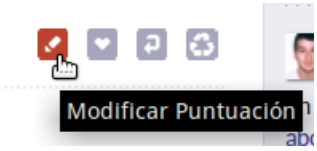


Ilustración 28 - Ejemplo de botón de modificación de nota

Ver puntuación. Relacionado con los dos casos de uso anteriores, es necesario que el alumno, y el resto de usuarios de BoloTweet, conozcan el *feedback* asociado a una determinada anotación. Por ello, tal y como se comenta en la sección *3.2.5*, se ha implementado un sistema que vincula a cada anotación su puntuación asociada, pudiendo visualizarse de forma inmediata por cualquier usuario. Este sistema recoge la puntuación asignada, así como el autor de la misma (*Ilustración 29*).

Ver desglose. En relación con el caso de uso anterior, y ya que *BoloTweet 2.0* soporta multiplicidad de profesores, como se detalla en la sección *3.3.2*, se ha implementado un sistema de desglose capaz de soportar múltiples puntuaciones. De esta manera, cuando más de un profesor califica un mismo *tweet*, se muestra la media de las calificaciones recibidas. No obstante, si el usuario desea conocer con qué nota le ha calificado cada profesor, tan sólo debe pulsar en el recuadro con la puntuación, y automáticamente aparecerá un cuadro desglosado indicando el autor de cada calificación respectivamente (*Ilustración 29*).

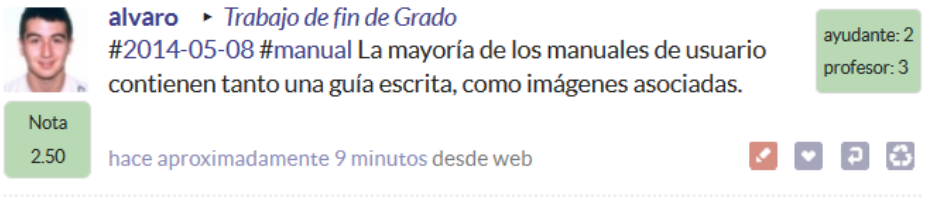


Ilustración 29 - Ejemplo de puntuación múltiple

Ver listado de puntuaciones. Al introducir un *feedback* a cada anotación, tal y como se explica en la sección *3.2.6*, se ha visto conveniente implementar un listado de calificaciones que permita a cualquier usuario conocer un ranking en tiempo real de los alumnos ordenados por mayor puntuación total. Este listado únicamente muestra los grupos de los cuales se es miembro, evitando así problemas de privacidad. Cada asignatura está contenida en un bloque expandible, y contiene un listado ordenado en el que se incluye el avatar del alumno, así como su nombre completo.

Ver listado completo. Relacionado con el caso de uso anterior, se ha implementado una ligera modificación del listado para los profesores, comentado en la sección *3.3.3*, de manera que se ha añadido a cada usuario de la lista su puntuación total, ya que es muy útil para la evaluación tanto de la clase en conjunto, pudiendo comparar las puntuaciones de los alumnos, como de manera individual.

Exportar calificaciones. Relacionado con el caso anterior, y tal y como se ha comentado en la sección *3.3.3*, se ha implementado un mecanismo de exportación de calificaciones restringido únicamente a profesores. A través de este mecanismo, al cual se accede desde el propio listado de puntuaciones, es posible generar para cada grupo un fichero CSV con el listado de usuarios ordenado por puntuación, junto con la calificación total asociada. El proceso de exportación completo, así como la configuración de los parámetros de exportación está detallado en el *Manual de Uso - Profesor*.

Ver tweets puntuados de un alumno. A partir de los listados de calificación generados, tal como se detalla en la sección *3.2.6*, se ha implementado un sistema de recopilación de *tweets* que ofrece la posibilidad de mostrar un listado con las anotaciones puntuadas vinculadas a un determinado usuario en un determinado grupo. De esta manera, es posible visualizar y revisar todas las anotaciones pertenecientes a un usuario determinado. Para acceder a esta opción, solamente se debe pulsar en el nombre del alumno que se desee a partir de los listados de calificación de alumnos vinculados a las asignaturas.

Estadísticas de alumno. En relación con el caso de uso anterior, y tal y como se comenta en la sección *3.3.3*, se ha implementado una ligera modificación de los listados de *tweets* puntuados vinculados a un determinado alumno, de manera que se ha incluido una sección con estadísticas de actividad. En esta sección, se han generado algunos datos como la puntuación total asociada al usuario, la puntuación media, el número de *tweets* totales, o el número de *tweets* puntuados.

Nombrar ayudante de grupo. Con el soporte de múltiples profesores por parte de *BoloTweet 2.0*, ha sido necesario implementar un sistema de nombramiento de ayudantes. De esta manera, los propios profesores pueden asociar nuevos profesores a un grupo sin necesidad de contactar con la administración. El proceso de asociación de profesores está explicado con detalle en el *Manual de Uso - Profesor*. No obstante, es importante destacar que únicamente puede nombrar ayudantes el profesor creador del grupo.

Ver profesores asociados a grupo. Al incorporar en *BoloTweet 2.0* el soporte de profesores con contexto, tal y como se detalla en la sección 3.1, se ha visto conveniente implementar un mecanismo que permita a un usuario conocer los profesores asociados a un determinado grupo. Por ello, en el perfil de cada grupo, se ha incorporado una nueva sección que recoge los profesores asociados a esa asignatura (*Ilustración 30*).

Ilustración 30 - Sección de profesores vinculados a un grupo

Distinguir actividad de profesor. Al implementar la distinción entre usuario y profesor, como se comenta en la sección 3, ha sido necesario incorporar una serie de cambios que permitan a cualquier usuario conocer de manera sencilla la actividad relacionada con profesores. Por ello, se ha decidido modificar su avatar y su perfil, obteniendo en el resultado de la *Ilustración 31*.

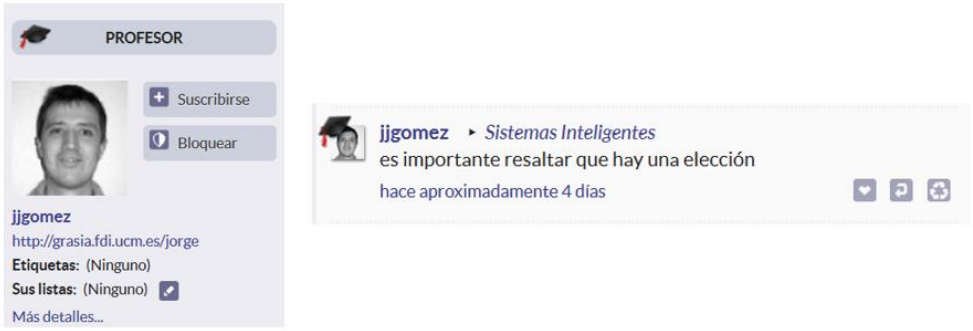


Ilustración 31 - Distinción de actividad de profesor

A continuación se muestran los casos de uso proporcionados por el desarrollo de los scripts de administración. El administrador es el responsable del mantenimiento del sistema y de crear la infraestructura de grupos que hace falta para que profesores y estudiantes hagan su parte.

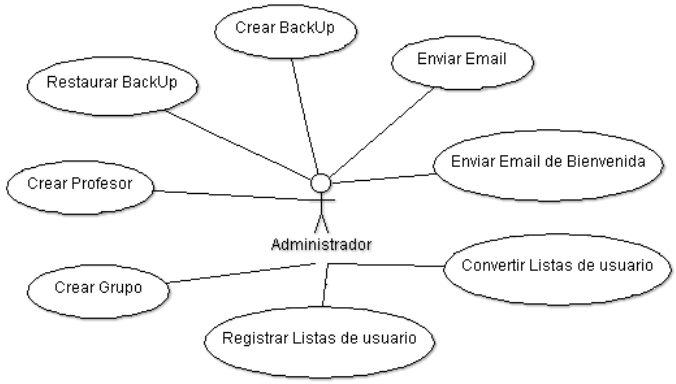


Ilustración 32. Casos de uso de un administrador

Crear Backup. Una de las carencias administrativas ha sido el sistema de copias de seguridad que se ha llevado a cabo al comienzo de uso de *BoloTweet*. Para solventar esto, se ha implementado un sistema de *backup* completamente automatizado. Este mecanismo se ha implementado en forma de script *Bash*, y únicamente puede ser utilizado por el administrador del sistema. Tras insertar la contraseña de super-usuario del sistema, y de la base de datos, automáticamente el script comienza a generar un fichero comprimido con las fuentes de *BoloTweet*, así como una copia completa de la base de datos. El proceso tarda unos pocos minutos, y consigue salvar el sistema en un fichero de apenas 40 megas. El proceso detallado de ejecución se puede encontrar en el *Manual de Uso - Administrador*.

Restaurar Backup. Para facilitar la administración del sistema, y relacionado con el caso de uso anterior, se ha implementado un script *Bash* capaz de automatizar por completo la tarea de restauración del *BoloTweet*. Únicamente puede ser utilizado por el administrador del sistema, ya que solicita la contraseña de súper-usuario y de la base de datos sobre la que se ejecuta el sistema. La restauración únicamente finalizará con éxito si el archivo de *Backup* ha sido generado por el script del caso de uso anterior. El proceso detallado de ejecución se muestra en el *Manual de Uso - Administrador*.

Enviar email. La necesidad de comunicación entre administrador y usuarios y la inexistencia en cuando a herramientas de este tipo, ha provocado la implementación de un script capaz de enviar *emails* automáticamente a cualquier usuario. Este script únicamente debe ser usado por el administrador del sistema. Su proceso de ejecución está detallado en el *Manual de Uso - Administrador*. Ha sido implementado de manera que ofrece 3 posibles opciones:

- *Envío individual.* El envío del correo se realiza a un único usuario.
- *Envío grupal.* El envío del correo electrónico se realiza a un grupo concreto.
- *Envío masivo.* El envío del correo se realiza a todos los miembros de *BoloTweet*.

Enviar email de bienvenida. En relación con el caso de uso anterior, y para proporcionar cierta información a los nuevos usuarios, ha sido necesario implementar un script capaz de enviar correos de bienvenida personalizados. Estos correos, generados dinámicamente, contienen un código de restauración de contraseña única, que permite a los usuarios recién registrados por la administración colocar la contraseña que deseen. Este script debe ser ejecutado por el administrador del sistema, y su uso y ejecución está detallada en el *Manual de Uso - Administrador*. Se ha implementado con 3 posibles funciones:

- *Envío individual.* El envío del correo se realiza a un único usuario.
- *Envío grupal.* El envío del correo electrónico se realiza a un grupo concreto.
- *Envío masivo.* El envío del correo se realiza a todos los miembros de *BoloTweet*.

Convertir listas de usuario. Para poder utilizar sistemas de micro-*blogging* en el ámbito educativo, es necesario disponer de herramientas que permitan gestionar una gran cantidad de usuarios. En este caso, para poder registrar listas de usuarios, se ha implementado un script *Bash* capaz de recibir una lista en formato *Excel* (*xls ó xlsx)*, y convertirla a un fichero CSV, para su posterior tratamiento. Este script solamente debe ser usado por el administrador, y su ejecución está detallada en el *Manual de Uso - Administrador*.

Registrar Listas de Usuarios. En relación con el script anterior, se ha implementado un sistema de registro automático de listas de usuarios. Este sistema se ha desarrollado en un script *Bash*, y es capaz de recorrer una lista (obtenida a través del script del caso de uso anterior) e ir registrando individualmente a los usuarios. Para poder elegir un nombre de usuario único de manera automática, se ha utilizado como *nickname* la primera parte del correo electrónico de cada usuario. En cuanto a la contraseña, se coloca una aleatoria, ya que posteriormente el usuario deberá modificarla a través del enlace recibido en el correo de bienvenida. Este script únicamente debe ser utilizado por el administrador bajo petición de profesores. Ofrece además la posibilidad de vincular los usuarios a un determinado grupo automáticamente al registrarse. El proceso detallado de ejecución se encuentra en el *Manual de Uso - Administrador*.

Crear Grupo. Una de las carencias en cuanto a la administración era la imposibilidad de la creación de grupos de manera automática. Tal y como se explica en la sección *3.1*, con la metodología propuesta cada grupo en *BoloTweet 2.0* representa a una asignatura, por lo que es necesario automatizar el proceso de creación de grupos. Por ello, se ha implementado un script capaz de crear un grupo y vincular como creador y administrador al profesor que lo haya solicitado. Este proceso únicamente puede ser llevado a cabo por el administrador del sistema. La ejecución del script se explica en el *Manual de Uso - Administrador*.

Crear profesor: Al incorporar el concepto de rangos a *BoloTweet*, tal y como se comenta en la sección *3.1*, ha sido necesario desarrollar mecanismos que permitan una cómoda gestión de roles. Por ello, se ha creado un script capaz de convertir automáticamente un usuario en profesor, y de vincularle como tal al grupo que se requiera. Este script contiene comprobaciones para soportar la creación de profesores ya existentes, por lo que puede usarse para asociar profesores a asignaturas, a pesar de que estos ya tengan los roles oportunos. También se ha implementado la opción contraria, por lo que es posible desvincular a un profesor de un grupo y quitarle todos los rangos. Únicamente puede ejecutarse por el administrador, y su funcionamiento está detallado en el *Manual de Uso - Administrador*.

4.2 Arquitectura de Status.Net

Para entender el funcionamiento de la mayoría del trabajo realizado sobre el sistema, es necesario comprender cómo funciona *StatusNet*. Como se comentó anteriormente, esta plataforma ofrece un software social orientado al micro-*blogging*, por lo que ahí radica su comportamiento. Sin embargo, tal y como se describe en su documentación, *Status.Net* está preparado para aumentar su funcionalidad a través de *plugins*.

Status.Net se comporta como una aplicación Web donde se procesan llamadas desde el navegador de un usuario. Los elementos que intervienen aparecen en la y realizan la siguiente serie de acciones:

1. Se produce una llamada *HTTP REQUEST*.
2. El Router obtiene la acción asociada a la *URL*.
3. Esa acción contiene hooks, que son utilizados por los *Plugins* a través de eventos.
4. La acción ejecuta un *prepare* para preparar los datos.
5. Posteriormente ejecuta el *handle*, donde se llevan a cabo todas las operaciones necesarias para la realización de la acción.
    a. Llamar a otras acciones.
    b. Mostrar una página con el contenido final.

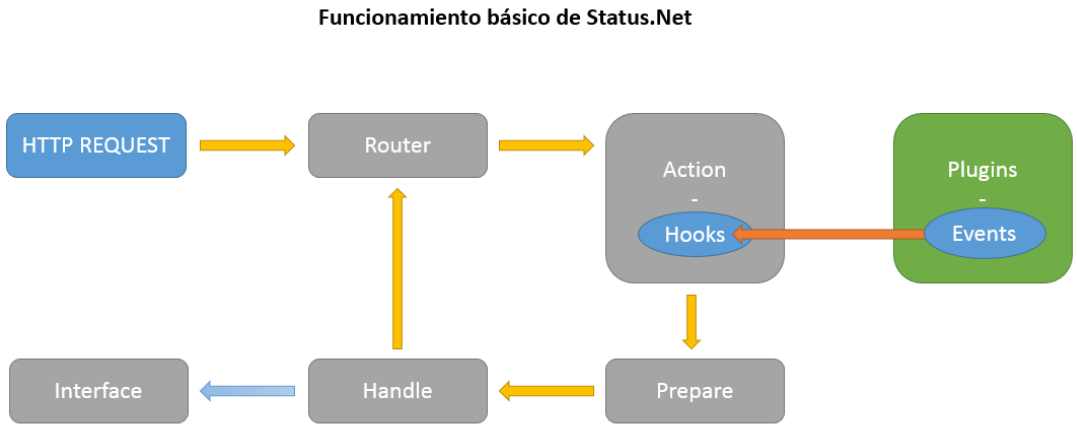


Ilustración 33 - Funcionamiento básico de StatusNet

Por ello, cuando se quiere expandir la funcionalidad de *Status.Net*, se puede hacer sobre el propio código fuente de la aplicación, en cuyo caso no se puede reaprovechar esa nueva funcionalidad si se actualizase el código original, o seguir el mecanismo de *plugins* que se propone. En tal caso, el desarrollo sería reutilizable y estaría desacoplado del núcleo de *Status.Net*.

Esto presenta dos ventajas principales. La primera es que permite activar y desactivar plugins de manera aislada, pudiendo elegir así de manera sencilla la colección de *plugins* activos en el sistema. La segunda consiste en aislar y desacoplar, como se ha comentado anteriormente, la funcionalidad del núcleo de *Status.Net*, lo que permite poder exportar, publicar y utilizar el plugin en diferentes instalaciones. Así pues, es pertinente el estudio del mecanismo de plugins de esta plataforma.

4.2.1 Desarrollo de Plugins

El desarrollo de un *plugin* implica definir una serie de elementos que son los que interactuarán con el núcleo de *StatusNet*. El elemento principal de interacción es el sistema de ganchos (*hooks*), a través de los cuales se puede alterar la información devuelta al usuario de forma transparente.

Estos ganchos normalmente son eventos del sistema, que permiten incluir elementos en un momento determinado, y en una zona concreta de la página. A través de la función *handle*, de *Event*, se definen *hooks* específicos. Cuando se ejecute la función *handle* vinculada a un *hook* (suelen ser métodos predefinidos que se pueden sobrescribir en el plugin), se buscarán funciones que realicen una llamada a ese enganche, y se irán ejecutando sucesivamente, permitiendo así realizar acciones en momentos concretos.

Un detalle importante, es que en ocasiones no existen *hooks* predefinidos para todos los lugares. No obstante, si se desea, se pueden crear de forma manual con la estructura mostrada anteriormente. Todos los *hooks* disponibles en el sistema vienen definidos y explicados en el fichero *EVENTS.txt*.

Una vez entendido el comportamiento básico de los *hooks*, es posible continuar explicando el funcionamiento de los *plugins*. En primer lugar, según la documentación proporcionada por StatusNet, los plugins propios de cada proyecto deben desarrollarse aislados del *núcleo*. Un buen lugar es la carpeta *local*, reservada para este tipo de desarrollos. A pesar de no indicarse una organización predeterminada para los *plugins*, en este proyecto se ha optado por utilizar una estructura de carpetas muy definida, permitiendo organizar e identificar de manera sencilla cada tipo de fichero.

A continuación se comenta brevemente la estructura empleada en el desarrollo de los plugins:

- *Fichero principal del Plugin*. Todo plugin debe disponer de un fichero principal que interactúe con el *core*. En este fichero, es donde deben realizarse todas las llamadas a *hooks* del sistema. El nombre de este archivo debe ser del tipo *xPlugin.php*, siendo *x* el nombre del *plugin*. Si el *plugin* contiene más archivos aparte del fichero principal, todos ellos deben cargarse utilizando el evento *onAutoLoad*. A pesar de poder importarlos manualmente en cada archivo con funciones como *require_once*, es conveniente realizar la carga a través de este evento, ya que éste se encargará de cargarlos únicamente cuando sean necesarios.

- Directorio CSS. Prácticamente todos los plugins medianamente complejos precisan de reglas CSS para mejorar su apariencia. Por ello, es necesario crear uno o varios archivos CSS, e incluir en ellos todas las reglas necesarias para la correcta visualización del plugin. Estos ficheros se deben alojar en la carpeta "css". En esta carpeta, también se incluirán todas las imágenes, iconos, fondos, o cualquier elemento relacionado con la apariencia del plugin.

- Directorio actions. Dependiendo de la complejidad del *plugin*, es muy común que se implementen acciones. Estas acciones responden a enlaces o a formularios visualizados por el usuario, y se encargan de realizar el procesamiento complejo para posteriormente actualizar el contenido de la página. Las acciones extienden de la clase *Action*, y deben implementar obligatoriamente las funciones *prepare* y *handle*. Cada acción va en un fichero distinto. Para que el sistema sepa cuándo ejecutar una determinada acción, es imprescindible vincularlas a una URL a través del evento *onRouterInitialized()*. Cuando el sistema recibe una determinada URL, a través de la clase *Router* se localiza la acción asociada. A continuación, se instancia la subclase de *Action* seleccionada, y se ejecutan sus dos métodos principales, *prepare* y *handle*.

- Directorio classes. Cuando un plugin necesita persistencia de datos personalizada, es necesario crear una clase por cada nueva tabla de la base de datos. Esas clases deberán contener una función *schemaDef*, en la que se definirá el esquema y la estructura completa de la tabla, y deberán extender de la clase *Managed_DataObject*. Es necesario que el esquema definido sea el correcto, ya que a partir de esta función se creará la tabla internamente. En este directorio, también se incluirán las clases encargadas de realizar una gran cantidad de operaciones que no es conveniente implementar directamente en acciones.

- Directorio lib. Esta carpeta alojará todos los archivos que representen a formularios, así como todas las librerías externas utilizadas en el plugin. En cuanto a los formularios, se deben desarrollar utilizando funciones predefinidas, y con una cierta metodología. Es necesario que este tipo de clases extiendan de la clase *Form* y utilicen los métodos proporcionados por esta.

- Directorio js. En esta carpeta se alojan todos los ficheros JavaScript propios del plugin. En este caso, los ficheros no tienen un formato de nombre específico, pero es importante que se vinculen al plugin utilizando el evento *onEndShowScripts* en el fichero principal. Lo más conveniente y común es implementar los scripts en forma de funciones, y llamarlas a través de eventos asociados a diferentes elementos de la página.

- Directorio scripts. En ocasiones, algunas funciones avanzadas realizadas en los plugins, como la actualización parcial de partes de la página, requiere el uso de marcos como *AJAX*. Sin embargo, para modificar este contenido, es necesario ejecutar un script *PHP*, que se encargue de contactar con la base de datos, o realizar las tareas oportunas, y posteriormente devolver un esqueleto *HTML* para incluir en el elemento que se está actualizando. Estos ficheros, utilizados por ejemplo en *Grades*, o en *Task*, se deben alojar en la carpeta *scripts*. No precisan de un formato específico, sin embargo, al tratarse de llamadas en segundo plano, es imprescindible importar manualmente todos los ficheros que se vayan a utilizar.

4.3 Diseño de los elementos de *Bolotweet 2.0*

En *Bolotweet 2.0* se requiere el desarrollo de *plugins* adicionales a los que vienen por defecto. Concretamente, se requiere el desarrollo de soporte para la nueva funcionalidad representada por los casos de uso *Grades*, *NotesPDF*, *Task* y *Guia*. Cada uno implica el desarrollo de un *plugin* específico.

Aparte de estos *plugins*, se añade otro adicional para mejorar el registro del sistema, *SecureRegistration*. Este último es una mejora necesaria dada las deficiencias que se han observado en cómo se registran los usuarios, ya que era sensible a ataques para registros masivos de anunciantes no invitados. En la sección *Tareas* del *Diario Beca de Colaboración*, se comentan algunos ataques sufridos durante el uso de *BoloTweet 2.0*, así como su solución.

En cada plugin se describe la estructura de clases que provee la funcionalidad requerida. En amarillo se representan aquellas clases que forman parte de *Status.Net* y con las que el plugin debe interaccionar. Los paquetes definidos en cada plugin obedecen a las distintas carpetas que se requieren para un plugin y que se listan en la sección *4.2.1*.

Aparte, se incluye el diseño de los scripts utilizados para administrar el sistema.

4.3.1 Diseño del plugin Grades

Este plugin desarrolla el caso de uso *Grades*. Este caso de uso abarca los casos de uso: *"ver puntuación", "nombrar ayudante de grupo", "gestionar puntuación", "ver profesores asociados a grupo", "distinguir actividad de profesor" y "ver listado de puntuaciones"*.

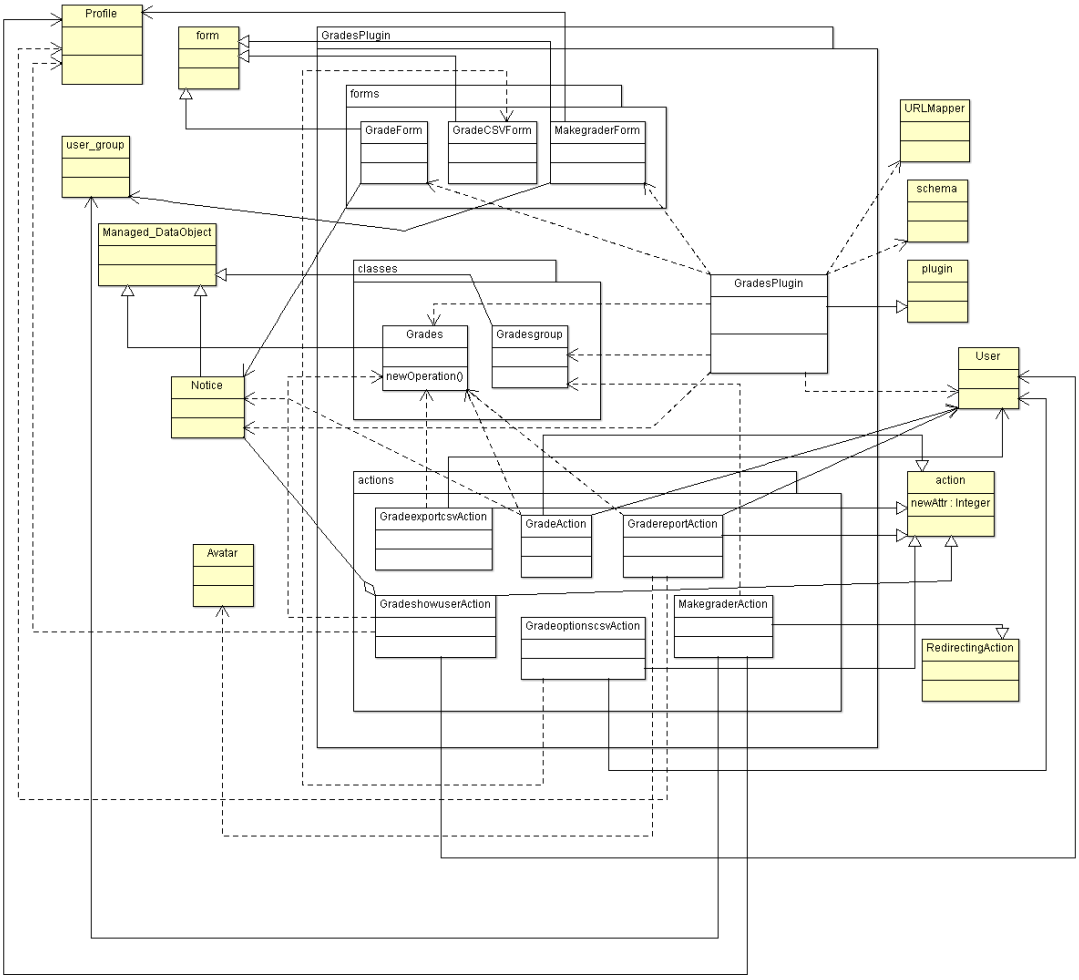


Ilustración 34 - Diagrama de clases de Grades

- ***Clase GradesPlugin*.** Esta clase representa el fichero principal del plugin. Extiende de la clase plugin, y en este caso se encarga de relacionar las funciones del plugin con el *core* a través de eventos. Se encarga además de cargar todos los ficheros del plugin, así como los ficheros CSS y los ficheros JavaScript. Utiliza la clase Schema para la definición de la estructura de tablas interna del plugin, y la clase URLMapper para asociar las acciones del plugin con sus correspondientes URL. Utiliza la clase Grades, para acceder a la función de definición de estructura interna, y a algunas de sus funciones de comunicación con la base de datos, necesarias para comprobar la validez de un grader o para obtener la puntuación y el profesor asociado a un *tweet*. Hace uso de la clase Gradesgroup, para acceder a la función de definición de la estructura interna de la tabla que representa, y para utilizar otras de sus funciones, necesarias para obtener los profesores asociados a un grupo, para validar si un profesor está asociado a un grupo, o para vincular un profesor con una asignatura. También utiliza la clase User, para obtener el usuario actual, y acceder a algunos de sus valores y funciones, por ejemplo para comprobar si tiene asociado un rol determinado. Otra clase de la que hace uso es Notice, en este caso para acceder a algunas de las funciones que permitan obtener datos acerca de las anotaciones, como el perfil del autor. Por último, se utilizan las clases MakegraderForm y GradeForm. La primera es necesaria para generar el formulario que se encarga de llamar a la acción que convierte un miembro de un grupo en profesor. La segunda clase, se utiliza para generar los formularios que corresponden a cada uno de los botones de puntuación asociados a las anotaciones.

- ***Clase gradeoptionscsvAction***. Esta clase, que extiende de la clase *action*, se encarga de generar la página de configuración de la exportación del listado de usuarios a un fichero CSV. Utiliza la clase User, almacenando una instancia de la clase como parámetro, para poder comprobar la validez del usuario (no vacío y logueado). También la utiliza para comprobar datos acerca del usuario, como si dispone del rango *grader*. Por último, utiliza la clase GradeCSVForm, para mostrar el formulario a través del cual el profesor puede seleccionar las distintas opciones de exportación.

- ***Clase gradeexportcsvAction***. Esta clase, que extiende de la clase *action*, se encarga de obtener los listados de puntuaciones relativos a un grupo, y de generar el fichero de exportación CSV con la configuración previa escogida por el profesor. Utiliza la clase User, almacenando una instancia de la clase como parámetro, para poder comprobar la validez del usuario (no vacío y logueado). También la utiliza para comprobar datos acerca del usuario, como si dispone del rango *grader*. Por último, utiliza la clase Grades para obtener los listados de puntuaciones asociados a un grupo concreto, necesarios para su posterior exportación.

- ***Clase gradeAction***. Esta clase, que extiende de la clase *action*, implementa todo el proceso relacionado con la calificación de las anotaciones. Utiliza la clase User, almacenando una instancia de la clase como parámetro, para poder comprobar la validez del usuario (no vacío y logueado). También la utiliza para comprobar datos acerca del usuario, como si dispone del rango *grader*. Utiliza la clase Grades para acceder a las funciones que le permiten registrar una nueva puntuación en la base de datos, actualizar un registro existente, o bien consultar la puntuación asociada a un *tweet*. Por último, utiliza la clase Notice, para obtener una instancia de la anotación asociada, validarla, y posteriormente utilizar algunos de sus valores como el ID, para generar un nuevo registro en la base de datos.

- ***Clase gradereportAction***. Esta clase, que extiende de la clase *action*, es la encargada de generar la página con el listado de calificaciones asociadas a cada grupo. Utiliza la clase User, almacenando una instancia de la clase como parámetro, para poder comprobar la validez del usuario (no vacío y logueado). También la utiliza para comprobar datos acerca del usuario, y distinguir así entre profesores y alumnos. Hace uso la clase Grades para poder obtener el listado de usuarios de cada grupo junto con su puntuación asociada. Por último, hace uso de la clase Profile, para obtener el perfil de cada usuario, y posteriormente utilizar la clase Avatar para colocar al lado de cada alumno su avatar personalizado.

- ***Clase gradeshowUserAction***. Esta clase, que extiende de la clase RedirectingAction, es la encargada de generar una página con el listado de anotaciones puntuadas asociadas a un determinado alumno en un grupo concreto. Por ello, esta clase almacena todo el conjunto de anotaciones filtradas, utilizando numerosas instancias de la clase Notice. Utiliza la clase User, almacenando una instancia de la clase como parámetro, para poder comprobar la validez del usuario (no vacío y logueado). También la utiliza para comprobar datos acerca del usuario, y distinguir así entre profesores y alumnos. Hace uso la clase Grades para poder obtener el conjunto de noticias puntuadas del alumno en un grupo concreto. Por último, también hace uso de la clase Profile, para obtener el perfil de cada usuario, obtener datos como el nombre completo, y para modificar el *aside* derecho de la página con el perfil del alumno en cuestión.

- ***Clase makegraderAction***. Esta clase, que extiende de la clase *action*, implementa todo el mecanismo necesario para convertir un miembro de un grupo en profesor del mismo. Utiliza la clase Profile, almacenando una instancia de la clase como parámetro, para poder comprobar la validez del usuario, y para posteriormente utilizar funciones sobre su perfil, tales como la comprobación de rango, o la asignación de nuevos roles. También hace uso de la clase User_group, almacenando una instancia de la misma como parámetro para comprobar su validez, y para poder validar si el usuario actual es el administrador del grupo. Por último, utiliza la clase Gradesgroup para vincular el usuario al grupo, y registrarlo en la base de datos.

- ***Clase Grades***. Esta clase, que extiende de Managed_DataObject, representa a la tabla Grades en la base de datos interna. Proporciona todas las funciones necesarias para realizar consultas sobre la tabla, e insertar y modificar registros sobre la misma. Esta tabla se encarga fundamentalmente de asociar *tweets* con sus respectivas puntuaciones.

- ***Clase Gradesgroup***. Esta clase, que extiende de Managed_DataObject, representa a la tabla Gradesgroup en la base de datos interna. Proporciona todas las funciones necesarias para realizar consultas sobre la tabla, e insertar y modificar registros sobre la misma. Esta tabla se encarga fundamentalmente de proporcionar un contexto a los profesores, asociándolos a asignaturas.

- ***Clase gradeForm.*** Esta clase, que extiende de Form, representa el formulario asociado a cada botón de calificación. Utiliza la clase Notice, de la que almacena una instancia como parámetro, ya que necesita utilizar algunos de sus valores para generar el formulario, y para posteriormente pasar información a la acción *GradeAction*.

- ***Clase gradeCSVForm.*** Esta clase, que extiende de Form, representa el formulario de configuración de las opciones de exportación de los listados de calificaciones.

- ***Clase MakegraderForm.*** Esta clase, que extiende de Form, representa el formulario a través del cual se convierte un miembro en profesor de un grupo. Utiliza la clase Profile, y User_Group, de las que almacena una instancia, ya que son necesarias para obtener valores y proporcionárselos a la acción *MakeGraderAction*.

4.3.2 Diseño del plugin NotesPDF

Este plugin desarrolla el caso de uso *NotesPDF*. Este caso de uso abarca los casos de uso: *"generar apuntes automáticos" y "generar apuntes personalizados"*.

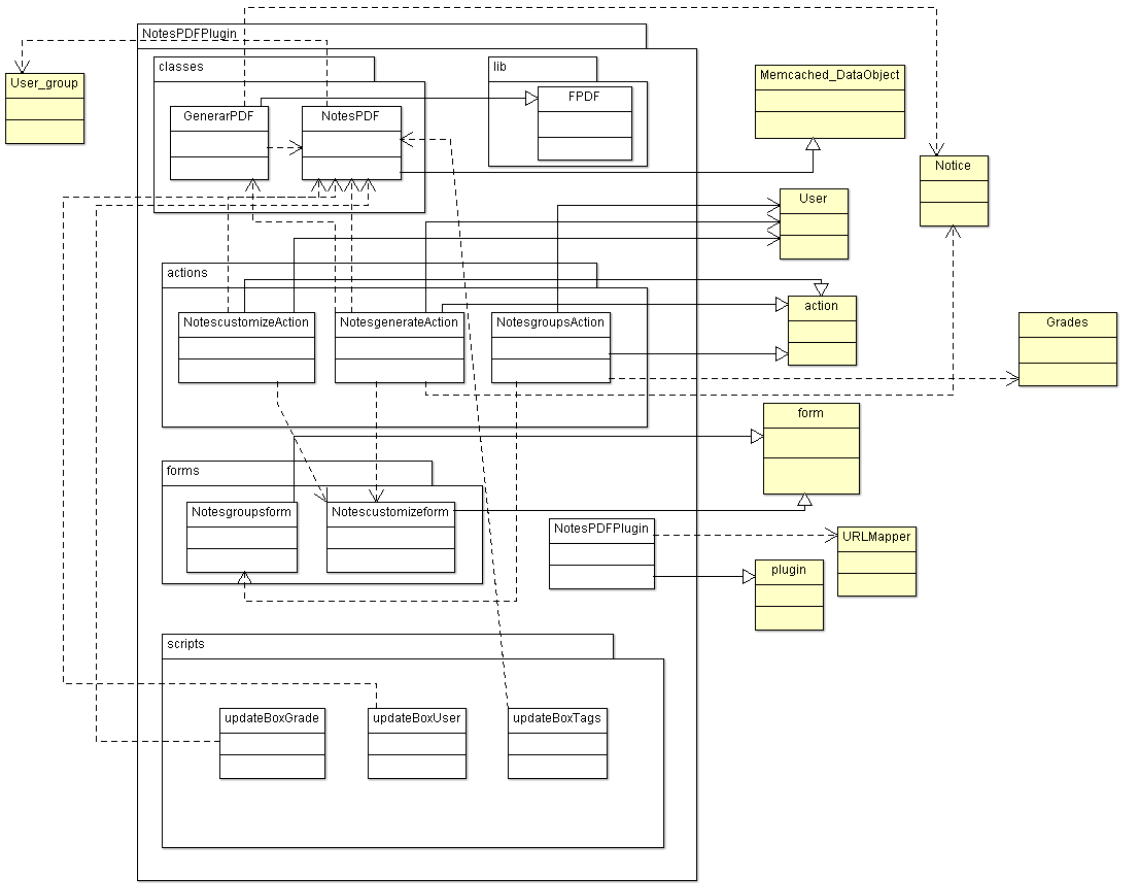


Ilustración 35 - Diagrama de clases de NotesPDF

- ***Clase NotesPDFPlugin.*** Esta clase representa el fichero principal del plugin. Extiende de la clase plugin, y en este caso se encarga de relacionar las funciones del plugin con el *core* a través de eventos. Se encarga además de cargar todos los ficheros del plugin, así como los ficheros CSS y los ficheros JavaScript. Utiliza la clase URLMapper para asociar las acciones del plugin con sus correspondientes URL.

- ***Clase notesGroupsAction***. Esta clase, que extiende de la clase *action*, se encarga de generar el listado de grupos para los cuales es posible generar tareas. Utiliza la clase User, almacenando una instancia de la clase como parámetro, para poder comprobar la validez del usuario (no vacío y logueado). También la utiliza para obtener datos acerca del usuario, como su ID, necesario para obtener sus grupos asociados. Utiliza la clase Grades para obtener los grupos con puntuaciones, ya que van a ser los únicos que permitan generar apuntes. Por último, hace uso de la clase NotesgroupsForm, que se encarga de generar el botón de comunicación entre esta acción y la de personalización de apuntes. Por último, utiliza la clase User_Group para obtener instancias de la clase, y poder obtener valores incluidos en los grupos.

- ***Clase notesCustomizeAction***. Esta clase, que extiende de la clase *action*, se encarga de generar la pantalla de personalización de apuntes vinculada a un grupo concreto. Utiliza la clase User, almacenando una instancia de la clase como parámetro, para poder comprobar la validez del usuario (no vacío y logueado), y para obtener algunos de sus datos, como el *nickname*, necesario en caso de redirección. Por otro lado, utiliza la clase NotesPDF, para obtener una instancia del grupo para el cual se están generando los apuntes, ya que es necesario obtener algunos datos como su nombre. Por último, se hace uso de la clase NotescustomizeForm, que se encarga de generar el formulario completo de elección de apuntes (automáticos y personalizados).

- ***Clase notesGenerateAction***. Esta clase, que extiende de la clase *action*, se encarga de generar la descarga de los apuntes seleccionados por el usuario. Utiliza la clase User, almacenando una instancia de la clase como parámetro, para poder comprobar la validez del usuario (no vacío y logueado), y para obtener algunos de sus datos, como el *nickname*, necesario en caso de redirección. Por otro lado, hace uso de la clase NotesPDF, para acceder a las funciones que permiten obtener los ID de las noticias seleccionadas por el usuario. Por otro lado, utiliza la clase Notice, para obtener instancias de todas las noticias filtradas, necesarias para la generación de los apuntes. Utiliza la clase GenerarPDF para la elaboración de apuntes y del posterior aviso de descarga. Por último, en caso de error, utiliza la clase NotescustomizeForm para volver a generar la pantalla de selección de apuntes.

- ***Clase NotesPDF***. Esta clase, que extiende de Memcached_DataObject, se encarga de proporcionar todas las funciones precisas para la comunicación con la base de datos, tales como la obtención de *tags*, los usuarios vinculados, o las puntuaciones de un determinado grupo, necesario para la carga del formulario de creación de apuntes personalizados. También proporciona otras funciones como la obtención de las noticias dada una serie de parámetros (usuario, tag y puntuación), o la obtención de noticias en el modo de apuntes automáticos. Utiliza la clase User_group para obtener una instancia del grupo para el cual se están generando los apuntes.

- ***Clase GenerarPDF***. Esta clase, que extiende de FPDF, contiene todo la implementación necesaria para la generación del fichero de apuntes final en formato PDF y su posterior descarga. Hace uso de la clase NotesPDF para obtener una instancia del grupo en cuestión, y por último, utiliza la clase Notice para acceder a funciones que permiten obtener datos de las anotaciones seleccionadas, como su autor.

- ***Clase notesGroupsForm.*** Esta clase, que extiende de Form, representa el formulario asociado a cada grupo en el listado de asignaturas con posibilidad de creación de apuntes.

- ***Clase notesCustomizeForm***. Esta clase, que extiende de Form, se encarga de generar el formulario de configuración de apuntes. Crea por tanto el apartado correspondiente a los apuntes automáticos, y a los personalizados.

- ***Clase UpdateBoxGrade***. Esta clase, utilizada por un script *Ajax*, se encarga de obtener las puntuaciones de las anotaciones vinculadas a un determinado grupo, atendiendo a dos parámetros adicionales, *usuario* y *tag*. Una vez obtenidas, las devuelve en un formato HTML para su posterior actualización en el cuadro de selección de puntuaciones correspondiente a los apuntes personalizados. Para obtener las puntuaciones, utiliza las funciones proporcionadas por la clase NotesPDF.

- ***Clase UpdateBoxUser***. Esta clase, utilizada por un script *Ajax*, se encarga de obtener los autores correspondientes a anotaciones puntuadas para un determinado grupo, atendiendo a dos parámetros adicionales, puntuación y *tag*. Una vez obtenidos, los devuelve en un formato HTML para su posterior actualización en el cuadro de selección de usuarios correspondiente a los apuntes personalizados. Para obtener los usuarios, utiliza las funciones proporcionadas por la clase *NotesPDF*.

- ***Clase updateBoxTag***. Esta clase, utilizada por un script *Ajax*, se encarga de obtener las etiquetas asociadas a anotaciones puntuadas para un determinado grupo, atendiendo a dos parámetros adicionales, usuario y puntuación. Una vez obtenidas, las devuelve en un formato HTML para su posterior actualización en el cuadro de selección de etiquetas correspondiente a los apuntes personalizados. Para obtener las etiquetas, utiliza las funciones proporcionadas por la clase NotesPDF.

- ***Clase FPDF.*** Esta clase representa a la librería *FPDF* y a todas las funciones necesarias para la generación y personalización de ficheros PDF.

4.3.3 Diseño del plugin para la Guía

Este plugin desarrolla el caso de uso *Guia*. Este caso de uso abarca el caso de uso: *"Descargar Manual de uso"*. Ver Ilustración 36.

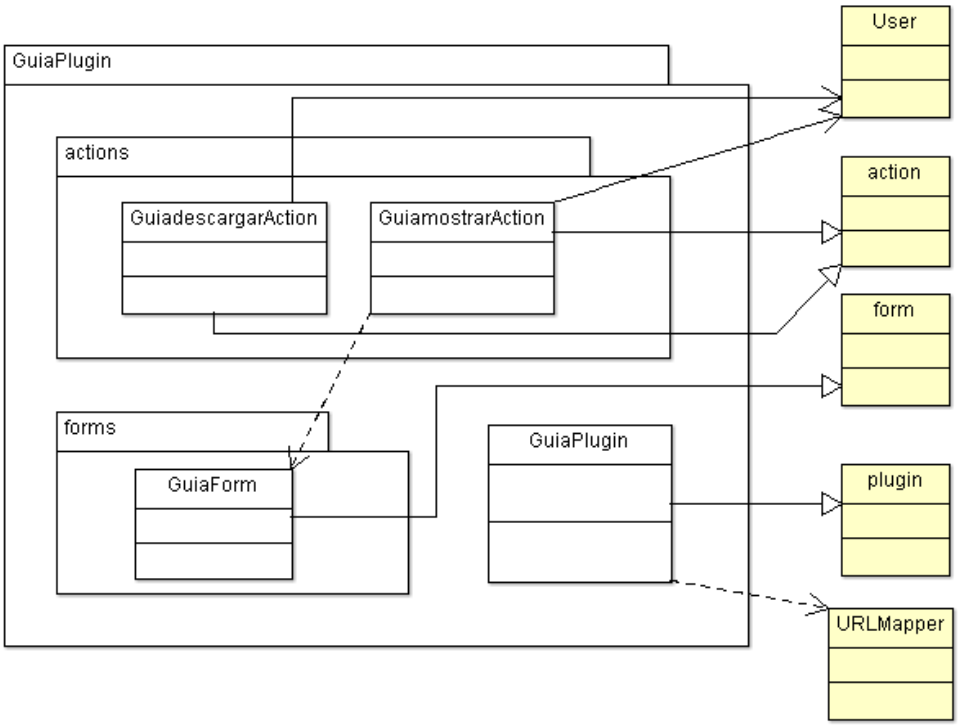


Ilustración 36 - Diagrama de clases de Guia

- ***Clase GuiaPlugin***. Esta clase representa el fichero principal del plugin. Extiende de la clase plugin, y en este caso se encarga de relacionar las funciones del plugin con el *core* a través de eventos. Se encarga además de cargar todos los ficheros del plugin, así como los ficheros CSS. Utiliza la clase URLMapper para asociar las acciones del plugin con sus correspondientes URL.

- ***Clase GuiamostrarAction***. Esta clase, que extiende de la clase *action*, representa a la acción encargada de recibir el enlace "Manual de Uso", mostrado en la página principal. Genera una página con información para la descarga del Manual. Utiliza la clase User, almacenando una instancia de la clase como parámetro, para poder comprobar la validez del usuario (no vacío y logueado). Por otro lado, utiliza la función GuiaForm para generar el formulario de descarga que se incluye en la página.

- ***Clase GuiadescargarAction***. Esta clase, que extiende de la clase *action*, representa a la acción encargada de generar la descarga del Manual de Uso. Utiliza la clase User, almacenando una instancia de la clase como parámetro, para poder comprobar la validez del usuario (no vacío y logueado), y en caso de ser necesario, abortar la descarga.

- ***Clase GuiaForm.*** Esta clase, que extiende de la clase *Form*, implementa el botón de descarga del manual a través de un formulario.

4.3.4 Diseño del plugin Secure Registration

Por temas administrativos, este plugin no está relacionado con ningún caso de uso. Los registros de usuario han pasado a ser un aspecto plenamente administrativo, por lo que se ha desactivado cualquier registro manual. No obstante, uno de los requisitos impuestos en la Beca de Colaboración comprendía el desarrollo de un sistema de registro seguro capaz de evitar registros no autorizados, por lo que se ha realizado su respectivo diseño e implementación.

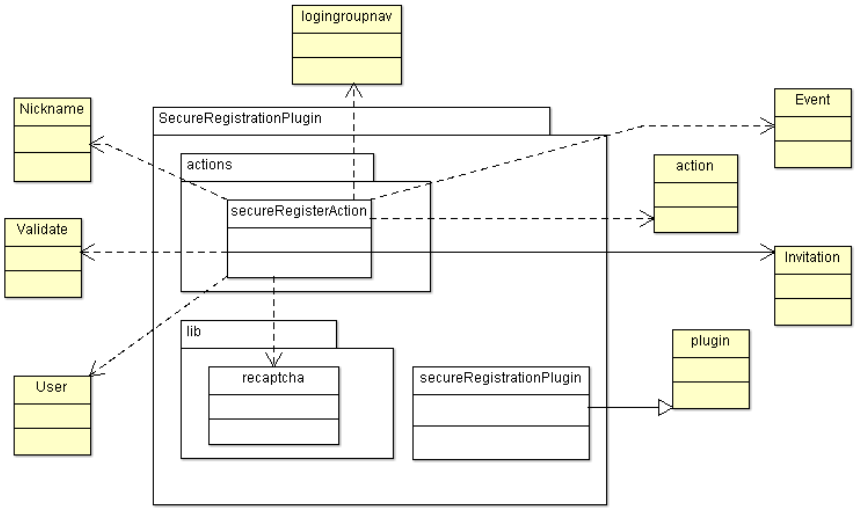


Ilustración 37 - Diagrama de clases de SecureRegistration

- ***Clase secureRegistrationPlugin*.** Esta clase representa el fichero principal del plugin. Extiende de la clase plugin, y en este caso se encarga de relacionar las funciones del plugin con el *core* a través de eventos. Se encarga además de cargar todos los ficheros del plugin, así como los ficheros CSS.

- ***Clase secureRegisterAction***. Esta clase, que extiende de la clase *action*, representa a la acción encargada de mostrar el formulario de registro seguro, así como de su tratamiento. Para generar el módulo de captcha, utiliza funciones proporcionadas por la clase *recaptcha*. Utiliza la clase User principalmente para registrar al usuario, y para utilizar funciones de validación del mismo. Por otro lado, utiliza la clase Invitation, almacenando una instancia de la misma como atributo, para poder tratar los registros basados en invitación. Por otro lado, utiliza las clases Validate y Nickname, para realizar comprobaciones de validación de los datos introducidos por el usuario. Por otro lado, también hace uso de la clase logingroupnav, para generar el menú de navegación de la página de registro, y por último utiliza la clase Event para proporcionar un *hook* a cualquier otro plugin que desee realizar acciones justo antes y después del proceso de registro.

- ***Clase recaptcha***. Esta clase representa a la librería *recaptcha*, un *wrapper* para PHP que se encarga de comunicarse con el servicio *Captcha* proporcionado por Google. Ofrece todas las funciones necesarias para la inclusión del módulo en cualquier página web.

4.3.5 Diseño del plugin Task

Este plugin desarrolla el caso de uso *Task*. Este caso de uso abarca los casos de uso: *"Resolución de tareas" y "Gestión de tareas".*

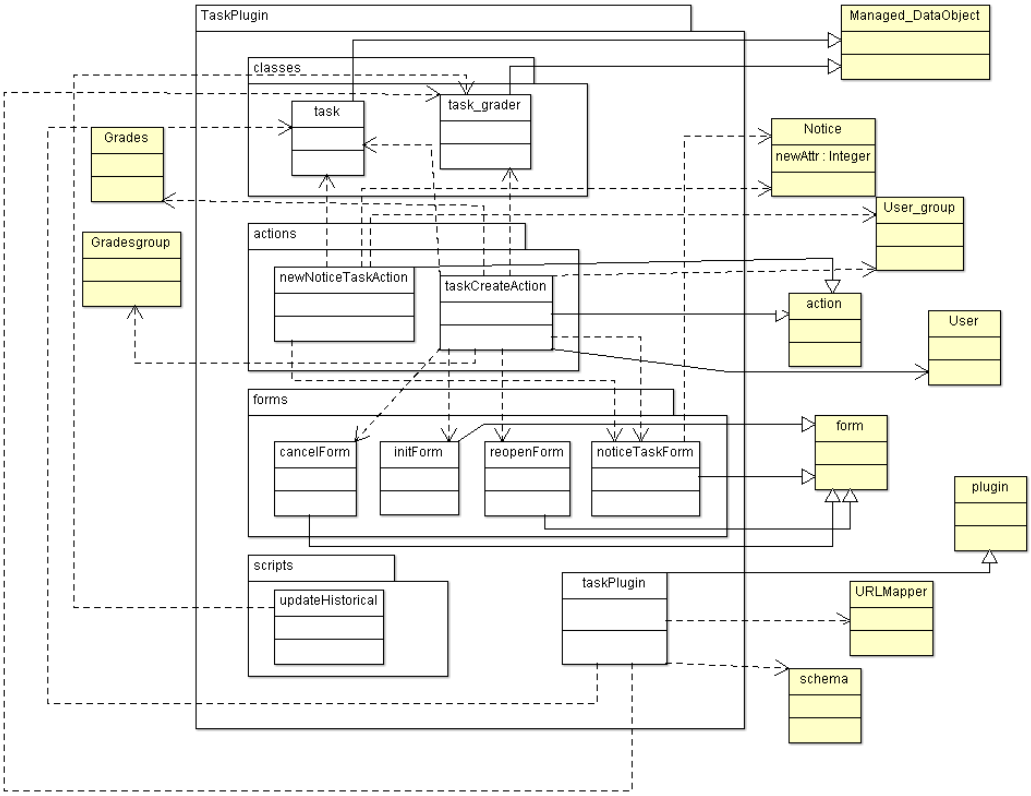


Ilustración 38 - Diagrama de clases de Task

- **Clase taskPlugin.** Esta clase representa el fichero principal del plugin. Extiende de la clase plugin, y en este caso se encarga de relacionar las funciones del plugin con el *core* a través de eventos. Se encarga además de cargar todos los ficheros del plugin, así como los ficheros CSS y los ficheros JavaScript. Utiliza la clase Schema para la definición de la estructura de tablas interna del plugin, y la clase URLMapper para asociar las acciones del plugin con sus correspondientes URL. Por otro lado, también hace uso funciones de la clase *Task* y Task_Grader para utilizar sus funciones de definición de tablas, y en el caso de Task, para obtener el número de tareas pendientes vinculadas a un usuario.

- ***Clase taskCreateAction***. Esta clase, que extiende de la clase *action*, se encarga de generar toda la funcionalidad ofrecida por el plugin Task. Trata la creación de tareas por parte de los profesores, la cancelación, modificación, reapertura e histórico de las mismas. Por otro lado también trata el rechazo y el envío de las tareas por parte de los alumnos. Utiliza la clase User, almacenando una instancia de la clase como parámetro, para poder comprobar la validez del usuario (no vacío y logueado). También la utiliza para obtener datos acerca del usuario, como el ID, y para comprobar su rango asociado. Utiliza la clase Task para acceder a las funciones

provistas por esta, como obtener tareas pendientes, o registrar, cancelar, reabrir o actualizar tarea. También hace uso de la clase Task_Grader, para acceder de nuevo a funciones proporcionadas por ella, como obtener histórico de tareas, o registrar, cancelar, reabrir, comprobar o actualizar tarea. Utiliza la clase Gradesgroup para obtener los grupos vinculados a un profesor, y la clase Grades para obtener los alumnos de un grupo. Utiliza las clases cancelForm, initForm, reopenForm y noticeTaskForm para crear los formularios asociados a cada una de esas acciones (cancelar, iniciar, reabrir, completar tarea…). Por último, utiliza la clase User_Group para obtener instancias de la clase, y poder obtener valores incluidos en los grupos.

- ***Clase newNoticeTaskAction***. Esta clase, que extiende de la clase *action*, se encarga de validar las tareas enviadas por los alumnos, y de registrarlas en la base de datos. Utiliza la clase User_group para obtener instancias del mismo, y así poder vincular anotaciones a grupos. Utiliza la clase Notice para realizar validaciones del contenido de la anotación, y finalmente para registrar el *tweet*. Hace uso de la clase Task para marcar como completada una tarea vinculada a un usuario en caso de que esta haya sido registrada. Por último, hace uso de la clase noticeTaskForm, para crear de nuevo el formulario que representa al cuadro de anotación en caso de surgir algún error en la validación del *tweet*.

- ***Clase Task***. Esta clase, que extiende de Managed_DataObject, representa a la tabla Task en la base de datos interna. Proporciona todas las funciones necesarias para comunicarse con la tabla de la base de datos, e insertar y modificar registros sobre la misma. Esta tabla se encarga de vincular a los alumnos con sus tareas.

- ***Clase Task_grader***. Esta clase, que extiende de Managed_DataObject, representa a la tabla Task_Grader en la base de datos interna. Proporciona todas las funciones necesarias para comunicarse con la tabla de la base de datos, e insertar y modificar registros sobre la misma. Esta tabla se encarga de vincular a los profesores con sus tareas.

- ***Clase cancelForm.*** Esta clase, que extiende de Form, representa el botón de cancelación de tarea a través de un formulario.

- ***Clase initForm.*** Esta clase, que extiende de Form, representa el formulario de creación de tareas para profesores.

- ***Clase reopenForm.*** Esta clase, que extiende de Form, representa el botón para reabrir una tarea a través de un formulario.

- ***Clase noticeTaskForm.*** Esta clase, que extiende de Form, representa el cuadro de anotaciones, así como todos sus componentes (caja de texto, caja de selección de grupo, contador de caracteres, botón de envío…) a través de un formulario.

- ***Clase updateHistorical***. Esta clase, utilizada por un script *Ajax*, se encarga de obtener las tareas asociadas a un profesor desde la base de datos, y devolver el histórico en formato HTML para su posterior actualización en la página. Utiliza la clase Task_Grader para obtener de la base de datos el histórico actual.

4.3.6 Diseño de scripts de soporte para administración

Estos scripts realizan los casos de uso asociados con la administración. Se responsabilizan de los casos de uso: "*Crear backup*", "*Restaurar backup*", "*Enviar email*", "*Enviar email de bienvenida*", "*Convertir listas de usuario*", "*Registrar listas de usuario*", "*Crear grupo*" y "*Crear profesor*".

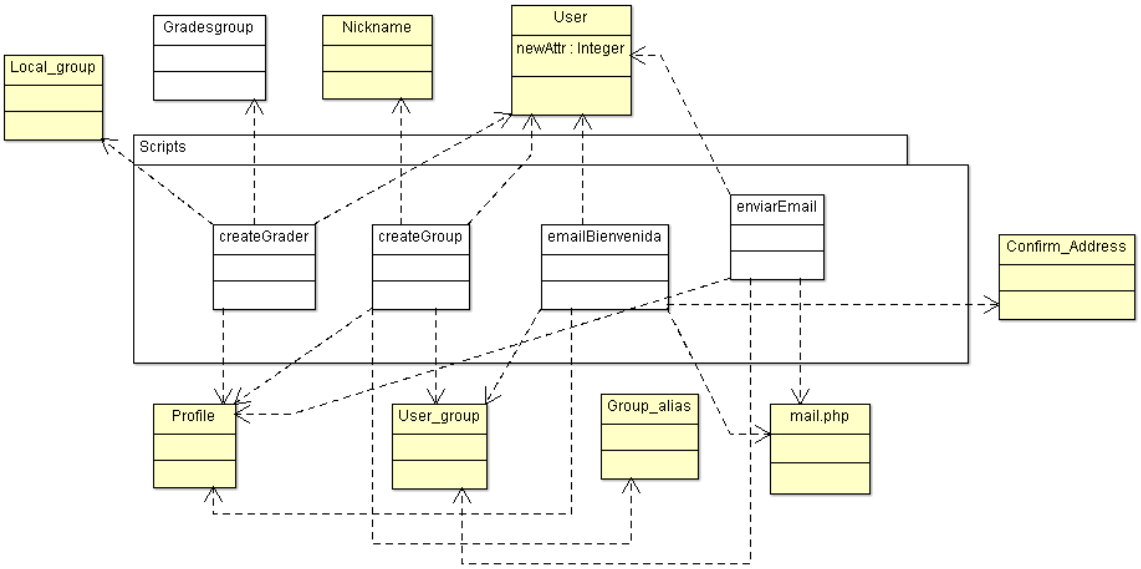


Ilustración 39 - Diagrama de clases de scripts de administración

- ***Clase createGrader.*** Esta clase representa al script PHP de creación de profesores. Utiliza la clase User para obtener el usuario pasado como parámetro, validarlo, y posteriormente utilizar sus datos. Con estos datos, se utiliza la clase Profile, para obtener el perfil asociado al usuario, y utilizar las funciones para asignarle rangos. Por otro lado, hace uso de la clase Local_group para obtener una instancia del grupo pasado como parámetro, y poder comprobar así su validez, ya que es necesario para vincular al profesor. Por último, utiliza la clase GradesGroup para vincular el usuario al grupo.

- ***Clase createGroup.*** Esta clase representa al script PHP de creación de grupos. Utiliza la clase User y Profile para obtener el usuario pasado como parámetro y validarlo, ya que será el que se vincule como creador y profesor del grupo. Por otro lado, se hace uso de Nickname y Group_alias para validar el *nick* del grupo, y evitar crear grupos con nombres o alias iguales. Por último, se utiliza User_Group para fijar las políticas del grupo que se va a crear, y posteriormente registrarlo.

- ***Clase emailBienvenida.*** Esta clase representa al script PHP de envío de correos de bienvenida. Utiliza la clase Profile para obtener los perfiles de todos los miembros de *BoloTweet*, en caso de escoger el envío masivo, y poder acceder a sus correos. Por otro lado, hace uso de la clase User, para obtener una instancia del usuario a enviar el correo, validarlo, y posteriormente obtener su dirección de correo, en caso de escoger un envío individual. También hace uso de la clase User_group, en el caso de que la opción escogida sea de envío a grupo. Usa la clase Confirm_addres para generar direcciones personalizadas a través de las

cuales poder modificar la contraseña. Por último, hace uso de la clase mail para el envío correo electrónico.

- ***Clase enviarEmail.*** Esta clase representa al script PHP de envío de correos. Utiliza la clase Profile para obtener los perfiles de todos los miembros de BoloTweet, en caso de escoger el envío masivo, y poder acceder a sus correos. Por otro lado, hace uso de la clase User, para obtener una instancia del usuario a enviar el correo, validarlo, y posteriormente obtener su dirección de correo, en caso de escoger un envío individual. También hace uso de la clase User_group, en el caso de que la opción escogida sea de envío a grupo. Por último, hace uso de la clase mail para el envío correo electrónico.

4.4 Implementación y despliegue

El desarrollo de *BoloTweet* se ha llevado a cabo sobre la plataforma *StatusNet*, ya existente. Este desarrollo ha consistido en la expansión de la plataforma a partir de *micro-aplicaciones*, implementadas utilizando el concepto de *plugins* propuesto por *StatusNet*, y detallado en la sección *4.2*. Los detalles de las modificaciones concretas realizadas se encuentran en el *Diario Beca de Colaboración*.

En cuanto a la implementación, para la parte web, se ha desarrollado en PHP, MySQL, CSS, HTML y JavaScript. También se ha hecho uso de algunos *frameworks* como JQuery o AJAX. Para la parte administrativa, se han desarrollado scripts utilizando PHP y *Bash*.

La implementación ha estado en todo momento dirigida por *issues*, y se ha llevado a cabo de forma incremental. Cada incremento y cada revisión de los casos de uso se gestionó a través de *issues*. Al tener solo un desarrollador, métodos de desarrollo más complejos resultaban inapropiados. Los *issues* se presentan en la *Tabla 1*, así como su correspondencia con los casos de uso. Al haberse realizado mejoras y resolución de *bugs* sobre el *core* de StatusNet, hay algunos *issues* no vinculados con ningún caso de uso.

Cada incremento estaba marcado por un conjunto de *issues* o por una actualización de *issues* desplegados en una *release* anterior. En esta filosofía, se definen los elementos de desarrollo en cada versión y se revisan los anteriores. El desarrollo es viable en este caso porque hay una arquitectura estable base, la de *Status.net*, y que el desarrollo respeta. Se crean fundamentalmente *plugins*, lo cual da lugar a *issues* bastante independientes, provocando un bajo acoplamiento durante el desarrollo. El nexo común de todos los elementos ha sido la interfaz de usuario.

En cuanto al control de cambios, se ha usado el sistema de control de versiones GIT, junto con un servicio de alojamiento web proporcionado por *BitBucket*. Sobre esto, se han llevado a cabo actividades regulares, como la realización de *commits* junto con su posterior subida al servidor *BitBucket* por cada modificación estable, consiguiendo así en todo momento una versión completamente funcional.

BitBucket también se ha utilizado para gestionar los *issues* y para discutir propuestas y soluciones para cada uno de ellos, precisando en ocasiones *tests* adicionales. Una vez iniciado el proceso de desarrollo asociado a un *issue*, se ha llevado a cabo un seguimiento del proceso de su resolución.

Issue	Status	Caso de Uso
#1: Puntuar mis propios twits	RESOLVED	Puntuar tweet
#2: Cambiar logo	RESOLVED	-
#3: opción delete account en cuenta	RESOLVED	-
#4: el cambio del idioma no va	RESOLVED	-
#5: editar puntuaciones	RESOLVED	Modificar puntuación
#6: Resaltar la puntuación del profesor	RESOLVED	Ver puntuación
#7: apuntes no se generan bien	RESOLVED	NotesPDF
#8: Hashtag asociado a twits	RESOLVED	Task
#9: opción "invite colleagues"	RESOLVED	-
#10: feeds piden contraseña	RESOLVED	-
#11: Marcar a los que sea GRADERS en la interfaz	RESOLVED	Distinguir actividad de profesor
#12: Ámbito de un GRADER	RESOLVED	Grades
#13: Combinación de grades	RESOLVED	Grades
#14: cambios varios en el documento de apuntes pdf	RESOLVED	NotesPDF
#15: Implementar registro seguro	RESOLVED	SecureRegistration
#16: Crear profesores desde interfaz	RESOLVED	Nombrar ayudante de grupo
#17: Al entrar por https, todo se ve fatal	RESOLVED	-
#18: El nuevo logo en los pdf no se ve	RESOLVED	NotesPDF
#19: Mala URL al ver un alumno	RESOLVED	-
#20: Escalabilidad de la parte derecha	RESOLVED	Grades
#21: Opciones Question y Event solo para algunos roles	RESOLVED	Grades
#22: Distinguir profesores en asignaturas	RESOLVED	Ver profesores asociados a grupo
#23: Falta logo versión móvil	RESOLVED	-
#24: Quitar logo búho en versión móvil	RESOLVED	-
#25: Letras Solapadas en versión móvil	RESOLVED	-
#26: Enlaces a 127.0.0.1 en varios sitios	RESOLVED	-
#27: Hacer un follow desaparecido	RESOLVED	-
#28: Mensajes Privados entre Suscriptores	RESOLVED	-
#29: Ver caracteres restantes	RESOLVED	Task
#30: Botones en versión móvil	RESOLVED	Grades
#31: Visualización Grade Reports	RESOLVED	Ver listado de puntuaciones Ver listado ampliado
#32: Desplazamiento de la página al evaluar	RESOLVED	Gestionar Puntuación
#33: Registro de tareas creadas	RESOLVED	Ver histórico
#34: correos para recordar contraseña expiran demasiado rápido	RESOLVED	-
#35: botones invisibles en interfaz de móvil	RESOLVED	-
#36: Enlaces en desarrollo apuntan a producción	RESOLVED	-
#37: Integridad Grades-Borrar Tweets	RESOLVED	Grades
#38: Localización Manual de Uso	RESOLVED	Descargar manual de uso
#39: Ver tareas pendientes en principal	RESOLVED	Ver número de tareas pendientes
#40: Resumen de Puntuaciones	RESOLVED	Ver desglose
#41: Exportar notas a formato csv	RESOLVED	Exportar listado de puntuaciones
#42: Meter un link en el grade reports a cada alumno	RESOLVED	Ver tweets puntuados de un alumno Estadísticas de alumno
#43: Separar el texto de los grupos en la columna de la izquierda	RESOLVED	-
#44: Crear mecanismo de comunicación admin-	RESOLVED	Enviar Email

usuarios		
#45: registro de usuarios automático	RESOLVED	Convertir Listas de usuario Registrar Listas de usuario
#46: crear graders automáticamente	RESOLVED	Crear Profesor
#47: Dar de alta asignaturas desde terminal	RESOLVED	Crear grupo
#48: Automatizar copia del sistema	RESOLVED	Crear BackUp Restaurar BackUp
#49: Información para nuevos usuarios	RESOLVED	Enviar Email de Bienvenida

Tabla 1- Listado de Issues de BoloTweet 2.0

Para cumplir con los requisitos de la *Beca de Colaboración* recibida, se ha ido escribiendo un diario en el que se ha ido comentando de manera detallada todas las tareas y los cambios realizados en BoloTweet. Este diario describe la implementación de los elementos aquí incluidos. El diario está en el anexo *Diario Beca de Colaboración*.

Por último, en cuanto al despliegue del sistema, y teniendo en cuenta que se ha puesto en marcha con usuarios reales, se ha utilizado la siguiente división de sistemas: local, desarrollo y producción. El proceso de desarrollo generalmente implicaba los siguientes pasos:

- Desarrollar localmente la solución al *issue*. Se prueba en la máquina del desarrollador siguiendo una configuración similar. Los cambios se van subiendo al repositorio *BitBucket* indicando en cada caso la naturaleza del cambio.
- Subir el desarrollo al servidor de desarrollo. Se trata de hacer disponible las mejoras en un servidor al que tienen acceso los profesores colaboradores en las pruebas. Cuando éstos validan los cambios se procede al siguiente paso.
- Subir el desarrollo a producción. Los alumnos acceden a las mejoras. Tanto alumnos como profesores no deben verse afectados por fallos en las mejoras aportadas. Por ello, este paso es el último y viene precedido de varias comprobaciones.

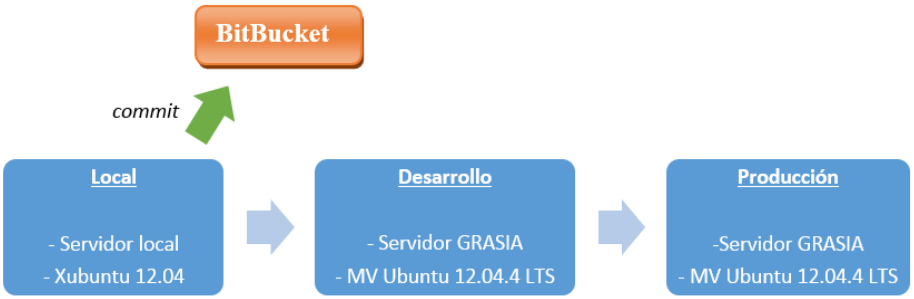


Ilustración 40 - Esquema de despliegue en BoloTweet 2.0

Por lo sensible de los cambios, cada mejora subida a producción debe ser avisada a los usuarios. Por ello, cada *release* desplegada en producción se ha comunicado a través del Campus Virtual mediante un foro. En cada aviso se ha mostrado una lista en forma de *release notes* con los cambios incluidos en la actualización. A este aviso, se une un correo para profesores con mayores detalles. La actualización del sistema dejaba inaccesible *BoloTweet* apenas 5 minutos. El proceso ha estado controlado en todo momento y no ha supuesto una molestia para los estudiantes.

A continuación se muestran las *release notes* de las 3 grandes actualizaciones realizadas.

Ilustración 41 - Primer aviso de actualización de BoloTweet 2.0

Release Notes:

- *Recuadro con las puntuaciones con tamaño fijo.*
- *Un profesor solamente puede puntuar tweets de sus asignaturas.*
- *Ahora se aceptan múltiples puntuaciones (de distintos profesores) para un mismo tweet.*
- *Se ha incluido un botón con forma de lápiz, en los botones de cada tweet, para modificar la nota asociada.*
- *Solucionada redirección al puntuar.*
- *Añadido apartado "Profesores" en el perfil de los grupos.*
- *Añadido enlace en la barra superior al Manual de Uso.*
- *Ahora los alumnos no pueden borrar sus tweets, únicamente pueden borrar los tweets propios los profesores.*
- *Corregidos bugs menores.*

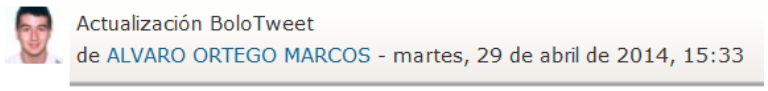


Ilustración 42 - Segundo aviso de actualización de BoloTweet 2.0

Release Notes:

- *Cambiado icono para modificar nota. [Profesores]*
- *Añadido desglose para tweets con puntuación múltiple al pulsar sobre la nota.*
- *Creados plugins para hacer un Backup y un Restore de toda la página automáticamente.*
- *Creado rol maxDeleter, con el cual se puede borrar cualquier tweet.*
- *Eventos y Encuestas restringidas sólo a Profesores.*
- *Añadido redondeo a dos decimales en las notas múltiples.*
- *Nombres de grupos en la columna izquierda cambiados por abreviaturas, y separados.*
- *Corregida visualización de la nota en los "replys".*
- *Disponible opción de apuntes personalizados.*
    - *Se ha habilitado la sección de apuntes personalizados, pudiendo escoger apuntes seleccionando un tag, un alumno, y una nota determinada.*
- *Modificado por completo mecanismo y apariencia de Grade Reports.*
    - *Solamente se pueden ver las puntuaciones para los grupos de los que eres miembro.*
    - *Al pulsar en el enlace de cada alumno, se visualizan los tweets de ese alumno en ese grupo.*
    - *Posibilidad de exportar los informes a CSV eligiendo separador y delimitador. [Profesores]*

- *Modificada visualización de autores en Apuntes, ahora se agrupan los tweets de cada autor.*

- *Creado concepto de Tareas.*
    - *Profesores pueden crear tareas para sus grupos.*
    - *Tareas pueden tener aparte del tag con la fecha, un tag personalizado extra.*
    - *Alumnos tendrán apartado con tareas pendientes, a través del cual podrán completarlas sin tener que poner el tag con la fecha, el tag opcional del profesor, ni el grupo al que va dirigido, se hace automáticamente.*
    - *Cada grupo tendrá un histórico de tareas. [Profesores]*
- *Corregidos bugs menores.*

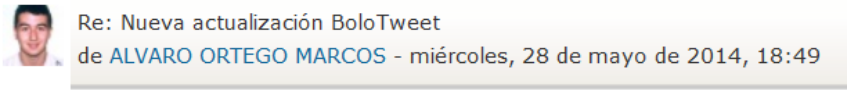


Ilustración 43 - Tercer aviso de actualización de BoloTweet 2.0

Release Notes:

- *Modificadas y añadidas traducciones al crear un nuevo grupo.*
- *Opción de crear nuevos grupos restringida sólo a profesores.*
- *Cambiado comportamiento de Encuestas, ahora el creador no tiene por qué contestarla.*
- *Solucionadas traducciones de Eventos, y añadidas nuevas.*
- *Corregido error de extensión al crear Eventos.*
- *Nuevas indicaciones al entrar a grupos privados o sin mensajes.*
- *Corregido bug interno al cancelar solicitudes de unión a grupo.*
- *Ahora en los listados de Grade Reports se muestran todos los alumnos, tengan o no puntuaciones.*
- *Al exportar listados a ficheros CSV ahora se incluyen todos los alumnos del grupo, no sólo los puntuados.*
- *Añadido refresco automático del número de tareas pendientes cada vez que se envía una nueva.*
- *Corregida traducción al guardar ajustes.*
- *Corregido bug interno al editar las opciones de un grupo.*
- *Ahora al acceder a los grupos se quedan los elementos de menú seleccionados.*
- *Creado script para crear grupo desde línea de comandos.*
- *Añadida vinculación automática cuando un profesor crea un nuevo grupo.*
- *Posibilidad de crear ayudantes del profesor desde la interfaz gráfica.*
- *Corregido bug al vincular usuarios a grupos privados automáticamente.*
- *Añadida información de salida a scripts existentes.*
- *Corregido bug interno al aprobar o rechazar usuarios pendientes en un grupo.*
- *Corregida visualización en versión móvil. Eliminado footer.*
- *Corregido PHPSESSID.*
- *Corregidos errores menores.*
- *Corregidas traducciones varias.*

4.5 Bugs

Durante el uso del sistema, se han ido encontrado algunos fallos existentes en la estructura interna de *StatusNet*. Para corregirlos, se ha tenido que trazar el comportamiento de los componentes relacionados, y finalmente se ha implementado la mejor solución sobre las fuentes del *core*. Como son errores generales producidos y generados por la plataforma, tras la finalización del proyecto, y una vez validados los cambios, se compartirán las soluciones en forma de *pull request* con el repositorio oficial de *StatusNet*. Esto significa que una vez sean aceptadas por los desarrolladores de *StatusNet*, pasarán a formar parte del *core* de la plataforma.

A continuación se muestran todos los errores internos encontrados durante el desarrollo de *BoloTweet*.

- Creación de grupo con nombre no válido.
- Usuarios con email provocan que el avatar desaparezca
- Error al cancelar una petición de unión a un grupo
- Error al aceptar una solicitud de unión a un grupo
- Error al modificar aliases en un grupo
- Al acceder a un grupo desde el menú lateral, no se queda seleccionado
- Corregido script *updateurls.php*

Su explicación detallada y solución, se puede encontrar en el apartado *Bugs* del *Diario Beca de Colaboración*.

4.6 Mejoras de código

El desarrollo de *Bolotweet 2.0* también ha requerido mejoras en el código de *Status.Net*. A medida que se ha ido añadiendo funcionalidad al sistema, se han tenido que ir realizando modificaciones sobre las fuentes originales para adaptarlas al comportamiento deseado de *BoloTweet 2.0*. En este apartado se encuentran todas aquellas que no tienen relación con *plugins* ni con errores internos del sistema.

Su explicación y solución está detallada en el apartado *Tareas* del *Diario Beca de Colaboración*.

- *Preparación entorno MySQL*
- *Enviar correos desde BoloTweet*
- *Deshabilitar la opción de eliminar cuenta para alumnos*
- *Cambiar idioma a Español*
- *Limpieza SPAM en servidor*
- *Quitar la opción Invite Colleagues*
- *Letras solapadas en la versión móvil*
- *Configurar BoloTweet con conexión obligatoria por HTTPs*
- *Eliminar FEEDS de BoloTweet*
- *Modificar y crear traducciones al Español*
- *Formulario cortado en carga de imágenes*
- *Error en el mensaje de la línea personal vacía*
- *En el panel de Administración el título aparece cortado*
- *Al crear una encuesta, el autor debe votar para ver los resultados*
- *Cualquier alumno puede crear una asignatura*

- *Error al enviar gran cantidad de eventos*
- *Cuando se accede a un grupo privado no se muestra ninguna información*
- *Cuando se accede a un grupo sin mensajes no se muestra ninguna información*
- *Corregidos errores menores*
- *Desarrollar un script para poder registrar listas de usuarios*
- *Escalabilidad de la parte derecha*
- *Exceso de tamaño al registrar listas de usuarios*
- *Desarrollar un Script para enviar correos de bienvenida*
- *Correo de bienvenida a los usuarios registrados*
- *Limpieza servidor de Desarrollo*
- *Alumnos no deben poder eliminar sus anotaciones*
- *Creación de rango para borrar cualquier mensaje*
- *Crear script para realizar copias de seguridad*
- *Crear script para restaurar la copia de seguridad*
- *Events y Polls solamente disponible para profesores*
- *Nombres de los grupos de la columna de la izquierda se ven muy juntos y mal*
- *Desarrollar un script para crear grupos*
- *Problema al vincular usuarios a grupos privados*
- *Añadir información de salida a algunos scripts*
- *Envío manual de correo de bienvenida utilizando registerUser.php*
- *Eliminar PHPSESSID de la URL*
- *Crear nuevos hooks onStartToolsLocalNav y onEndToolsLocalNav*
- *Cambiar el logo de StatusNet*
- *Añadir logo versión móvil*
- *Cambiar tema por defecto*

4.7 Conclusiones

El desarrollo de *BoloTweet 2.0* se ha llevado a cabo sobre la plataforma ya existente, *StatusNet*, utilizando el sistema de *plugins* y *hooks* propuesto en su documentación. La contribución fundamental de *BoloTweet 2.0* reside en el desarrollo de 5 *plugins* independientes, *Grades*, *NotesPDF*, *Guia*, *SecureRegistration* y *Task*, cada uno de ellos diseñado para satisfacer los casos de uso propuestos en la sección *4.1*. No obstante, para lograr el comportamiento actual de *BoloTweet 2.0*, se han tenido que corregir errores internos descubiertos en la plataforma *StatusNet*, así como mejoras adicionales implementadas sobre el *core* de la plataforma, y que no tienen relación directa ni con *bugs*, ni con *plugins*. También ha sido necesario implementar un conjunto de herramientas en forma de *scripts*, para poder gestionar y administrar tanto el sistema como sus usuarios de una manera cómoda y automatizada.

En cuanto a la implementación se ha desarrollado en PHP, MySQL, CSS, HTML, JavaScript, JQuery, Ajax y *Bash*. Se ha llevado a cabo una implementación incremental dirigida por *issues*, y controlada a través del SCM *GIT* junto con el servicio de alojamiento web, *BitBucket*. Teniendo en cuenta que *BoloTwet 2.0* se ha puesto en marcha con usuarios reales, ha sido necesario utilizar un despliegue dividido en sistemas: local, desarrollo y producción. Toda actualización realizada en *producción* se ha realizado con previo aviso a los usuarios a través de un foro en el CV.

5 Evaluación del trabajo

En esta sección se realiza una valoración del trabajo realizado. Primero, una descripción del alcance del mismo usando métricas de ingeniería del software. El objetivo de esta primera sección es explicar cuánto trabajo se ha invertido. La segunda sección explica cómo los estudiantes han usado el sistema. Esta sección justifica que ha habido un uso real coherente con lo que se esperaba del método que propone *Bolotweet*. En concreto, que se usa durante el horario de clase, principalmente. La tercera sección incluye una valoración de los estudiantes de la aplicación. Esta valoración es relevante para justificar si los estudiantes aceptan *Bolotweet 2.0* y si opinan que el sistema es fácil de usar o no.

5.1 Valoración del esfuerzo realizado

Como se ha comentado en varias secciones del documento, *BoloTweet* se ha construido sobre una base ya desarrollada, proporcionada por *StatusNet*.

Sobre el sistema base, se han añadido una serie de *plugins* que han proporcionado nuevas funcionalidades al sistema. Estos *plugins* se han ido desarrollando a lo largo del proyecto para satisfacer las demandas planteadas en cada momento.

- *Grades*
- *NotesPDF*
- *Task*
- *Guia*
- *SecureRegistration*

Tal y como aconsejan en la *StatusNet Wiki* [36], todos estos plugins se han desarrollado en la carpeta *local*, por lo que todo su contenido es contribución propia.

Por otro lado, también ha sido necesario desarrollar herramientas para la administración del sistema, inexistentes por defecto en *StatusNet*. Todas estas herramientas se han implementado en forma de scripts, bien en *bash*, o en *PHP*, y se pueden encontrar en la carpeta *scripts*.

Los scripts que se han desarrollado por completo son:

- *backupBT.sh*
- *restoreBT.sh*
- *emailBienvenida.php*
- *enviarEmail.php*
- *createGrader.php*
- *createGroup.php*
- *convertirCSV.sh*
- *registrarListas.sh*

Algunos scripts preexistentes en *StatusNet*, han sido modificados para mejorar su funcionalidad:

- *registeruser.php*
- *joingroup.php*

- *updateurl.php*
- *leavegroup.php*
- *makegroupadmin.php*

También se ha contribuido solucionando errores internos del sistema (*bugs*), hasta un total de 7, y modificando archivos internos del *core* para lograr la funcionalidad deseada (creación de nuevos *hooks*, traducciones, activación HTTPs, modificaciones en la apariencia móvil…).

Para poder contabilizar de manera física el trabajo realizado, se ha utilizado el software de medición *CLOC* [35]. Esta herramienta cuenta las líneas en blanco, las líneas de comentario, y las líneas de código en un determinado directorio.

Para poder estimar de manera precisa la contribución aportada, se ha aislado por un lado la carpeta local, con todos los *plugins*, y por otro lado los scripts de administración desarrollados.

El resto de scripts, o de ficheros modificados durante el trabajo, no se han contabilizado para la estimación, ya que al no haberse desarrollado por completo, y no poder definir trozos específicos de código, la estimación resultante no sería realista.

Además, las librerías utilizadas en algunos plugins, como la librería *FPDF*, en el plugin *NotesPDF*, o *reCAPTCHA*, en *SecureRegistration*, también se han eliminado, para obtener una estimación lo más fiel posible a la realidad.

A continuación se comentan los resultados obtenidos.

Language	Files	Blank	Comment	Code
PHP	50	1487	1609	4182
CSS	6	207	42	746
Bourne Shell	4	61	68	187
JavaScript	3	105	22	140
SUM	63	1860	1741	5255

Por otro lado, también es interesante estimar el trabajo realizado en base a la herramienta de control de versiones *GIT*. Para obtener análisis acerca del repositorio y todas las acciones desarrolladas en él, se ha utilizado la herramienta *gitStats* [34], y se han generado los resultados mostrados a lo largo de esta sección.

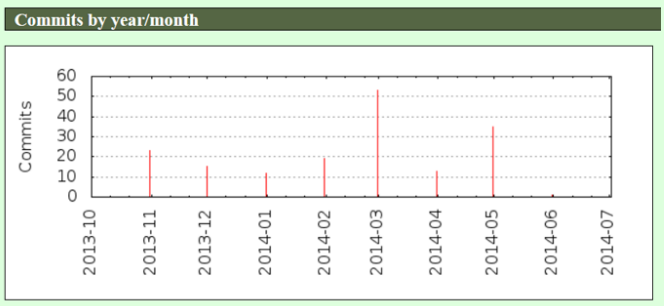
El trabajo realizado sobre el repositorio es relevante pues muestra de manera precisa cuál ha sido la aportación del trabajo, así como el sistema y la organización del desarrollo. En la *Ilustración 44*, se pueden ver datos generales acerca de la actividad del repositorio. Destacar los 200 días de desarrollo, o los 171 *commits* totales. Cada *commit* identifica a un cambio estable que se ha desarrollado e implementado sobre el sistema. Esto es importante, ya que en ningún momento se han subido modificaciones con errores, o que no funcionaban correctamente. Por ello, antes de subir al repositorio remoto una modificación, ha sido implementada y testeada en local durante días. Cada vez que se terminaba un cambio y se comprobaba su correcto funcionamiento, se ha subido al servidor a través de un *commit*.

Como se puede ver en las estadísticas, se ha interaccionado con *BitBucket* un total de 59 días, correspondientes a aquellos en los que había algún cambio estable y testeado. Por ello, cada uno de esos días tiene una media de casi 3 *commits*.

Project name:
bolotweet
Generated:
2014-06-05 15:48:02 (in 37 seconds)
Generator:
GitStats (version 2011.11.08), git version 1.7.9.5, gnuplot 4.4 patchlevel 3
Report Period:
2013-11-17 17:27:39 to 2014-06-05 15:42:40
Age:
200 days, 59 active days (29.50%)
Total Files:
4472
Total Lines of Code:
2048 (20342 added, 18294 removed)
Total Commits:
171 (average 2.9 commits per active day, 0.9 per all days)
Authors:
1 (average 171.0 commits per author)

Ilustración 44 - Información general del control de versiones

En la *Ilustración 45*, se muestra una gráfica de *commits* por mes, alcanzando el máximo nivel de desarrollo en la última parte del proyecto, desde el mes de marzo. Esto es debido a que en este punto, se terminaron de resolver problemas generales, y se comenzaron a desarrollar con más continuidad los *plugins*.



Month	Commits	Lines added	Lines removed
2014-06	1	176	6
2014-05	35	6707	6563
2014-04	13	2059	181
2014-03	53	1518	395
2014-02	19	234	60
2014-01	12	621	31
2013-12	15	901	599
2013-11	23	8126	10459

Ilustración 45 - Estadísticas de commits por meses

Por último, en la *Ilustración 46*, se puede ver el progreso de *commits* por autor a lo largo del periodo de desarrollo. En este caso, como el desarrollo ha sido individual, únicamente se muestra la evolución de los commits para un solo autor, en este caso hasta llegar a los 171.

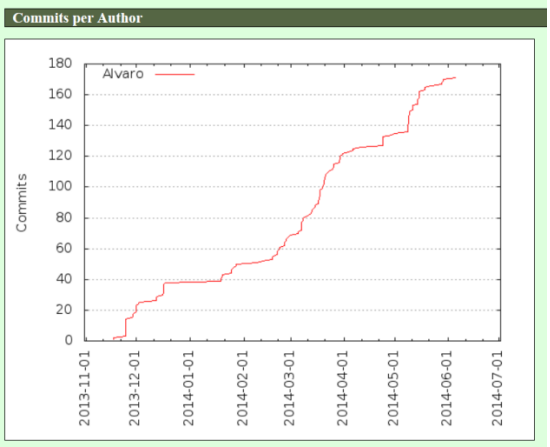


Ilustración 46 - Progreso de commits

5.2 Valoración de uso del sistema

El sistema se ha usado en cuatro asignaturas: DASI (Desarrollo de Aplicaciones y Sistemas Inteligentes), PEVOL (Programación Evolutiva), SSII (Sistemas Inteligentes), y SWEB (Sistemas Web). Los estudiantes matriculados en cada una eran 16, 45, 38 y 42 respectivamente.

En esta sección se muestra la actividad recibida por el sistema desde su puesta en marcha con usuarios reales. Ya que *StatusNet* no proporciona ninguna herramienta de análisis de información para poder obtener datos acerca de la actividad, se ha tenido que recopilar e interpretar la información directamente desde la base de datos. El resultado es un informe completo sobre la actividad de los usuarios en *BoloTweet 2.0*.

- **Período de uso** [Utilizando la primera y la última fecha de *tweets*]

  Del 27 de Febrero al 4 de Junio *(Calculado con timeanddate.com)*

  - Días totales: 98
  - Días naturales (sin fines de semana ni festivos): 68
- **Número de asignaturas: 4**
- **Número de usuarios registrados : 125**
- **Número de *tweets* totales: 1127**
- **Número total de *tweets* de usuarios: 861**
- **Número de *tweets m*arcados como favoritos: 52**
- **Número de puntuaciones emitidas por profesores: 587**
- **Número de *tags* únicos: 147**
- **Número de *tags* totales: 1412**
- **Tareas creadas por profesores: 5**
- **Tareas completadas por alumnos: 70**
- **Promedio de tweets:**
  - Días totales: 861/98 = **8.79***tweets*/día
  - Días lectivos: 861/68 = **12.66***tweets*/día
- **Número de anotaciones por grupo:**

DASI	**100**
PEVOL	**174**
SWEB	**213**
SSII	**425**

La evolución de uso del sistema ha sido estudiada también. Para ello, la consulta realizada sobre la base de datos se ha exportado a un fichero TXT, en el que se incluye una lista con la frecuencia de envío de *tweets* por día. Esa lista se ha importado en Excel, y se ha generado la siguiente gráfica *(ver Ilustración 47).*

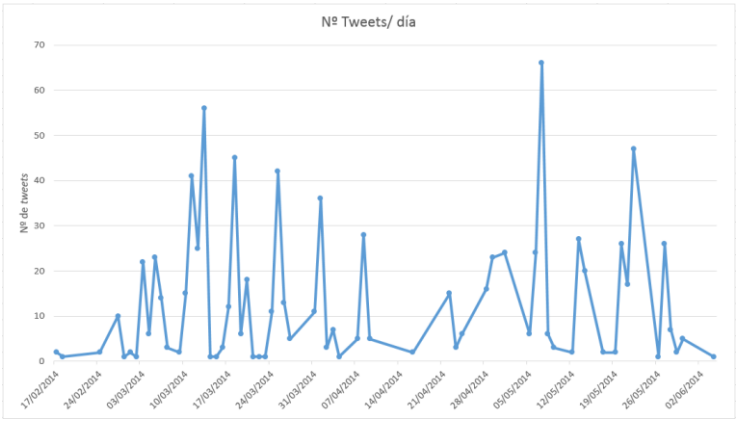


Ilustración 47 - Evolución de uso de BoloTweet 2.0

Como se puede observar en la gráfica, hay un gran uso del sistema entre los meses de febrero a mayo, existiendo una decaída en el mes de abril que coincide plenamente con las vacaciones de semana santa.

Por otro lado, observando la *Tabla 2*, es interesante destacar algunos aspectos de la evolución de uso del sistema.

Fecha	**Nº**	**Fecha**	**Nº**	**Fecha**	**Nº**
17/02/2014	2	16/03/2014	3	01/05/2014	**24**
18/02/2014	1	17/03/2014	**12**	05/05/2014	6
24/02/2014	2	18/03/2014	**45**	06/05/2014	**24**
27/02/2014	**10**	19/03/2014	6	07/05/2014	**66**
28/02/2014	1	20/03/2014	**18**	08/05/2014	6
01/03/2014	2	21/03/2014	1	09/05/2014	3
02/03/2014	1	22/03/2014	1	12/05/2014	2
03/03/2014	**22**	23/03/2014	1	13/05/2014	**27**
04/03/2014	6	24/03/2014	11	14/05/2014	**20**
05/03/2014	**23**	25/03/2014	**42**	17/05/2014	2
06/03/2014	**14**	26/03/2014	**13**	19/05/2014	2
07/03/2014	3	27/03/2014	5	20/05/2014	**26**
09/03/2014	2	31/03/2014	**11**	21/05/2014	17
10/03/2014	**15**	01/04/2014	**36**	22/05/2014	**47**
11/03/2014	**41**	02/04/2014	3	26/05/2014	1
12/03/2014	**25**	03/04/2014	7	27/05/2014	**26**
13/03/2014	**56**	04/04/2014	1	28/05/2014	7
14/03/2014	1	07/04/2014	5	29/05/2014	2
15/03/2014	1	28/04/2014	**16**	30/05/2014	5
24/04/2014	6	29/04/2014	**23**	04/06/2014	1

Tabla 2 - Frecuencia de uso de BoloTweet 2.0

En primer lugar, todas las anotaciones existentes se han enviado en un total de 65 días. Mirando el número de envíos por día, en 27 ocasiones se puede deducir que los *tweets* se han generado desde clase, ya que coinciden las horas pico con el horario de las clases. Esto confirma el uso del sistema en la línea del proyecto: durante las horas de clase.

El resto de días, las anotaciones han sido aisladas, con envíos desde 1 hasta 6 *tweets*. Esto indica que muy probablemente se hayan realizado en horario externo a las clases, por cuenta propia de los alumnos. De esto se puede concluir, que además del envío ordinario de clase, los alumnos participan y aportan sobre otras anotaciones de los compañeros, lo que a su vez revela que una buena cantidad de alumnos utiliza la herramienta fuera del horario académico, y se preocupa por leer e interactuar con las ideas plasmadas en *BoloTweet*.

En la *Ilustración 48*, se puede ver como el uso de la herramienta ha ido aumentando con el tiempo, desde febrero hasta mayo.

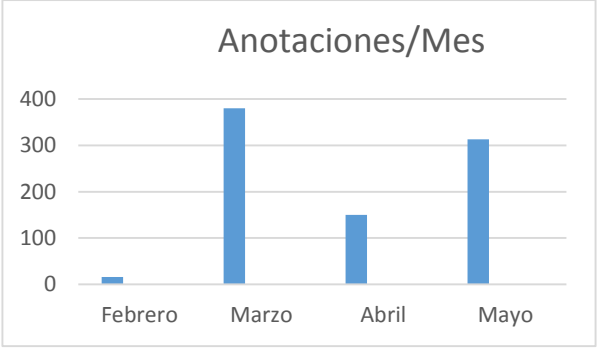


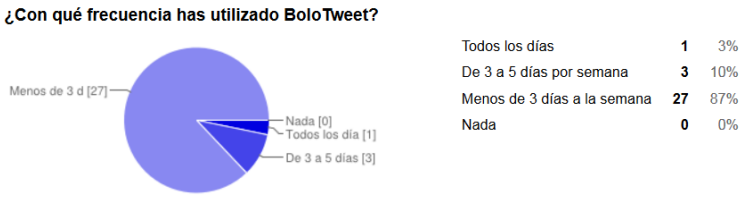
Ilustración 48 - Progreso de uso de BoloTweet

El valor de los tweets no se ha valorado apropiadamente. Entre las respuestas recogidas a través de la herramienta, se pueden encontrar *tweets* interesantes (*Ilustración 49*) relacionados con las clases, y que son un recurso de conocimiento para al resto de compañeros, ya que han sido puntuados con la máxima calificación.

Ilustración 49 - Ejemplo anotaciones interesantes

5.3 Valoración de los estudiantes

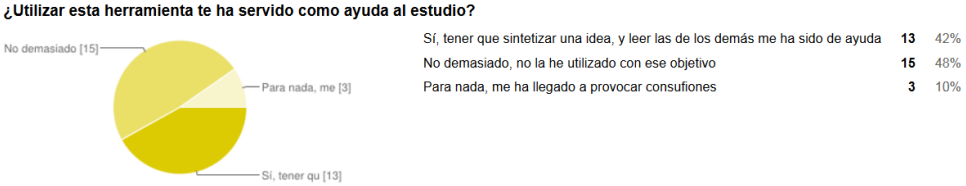
Uno de los aspectos más importantes en cuanto al desarrollo y la implantación de la herramienta, es la opinión de los propios usuarios. Para ello, se ha usado una encuesta online anónima utilizando un formulario de Google Drive, y se han obtenido los siguientes resultados:



Como se puede observar, el uso medio de la aplicación es de 0 a 3 días por semana. Es un uso lógico, ya que coincide con el número de días en los que se imparte una asignatura por semana.

Considerando que *BoloTweet* se comenzó a utilizar a mediados de Febrero del año 2014, y que se ha mantenido una línea de uso y desarrollo paralela, por lo que no han podido utilizar al completo el potencial de la herramienta, los resultados son correctos.

El manejo de la herramienta es uno de los puntos mejor valorados. Esto indica que la metodología aplicada, así como el comportamiento proporcionado a través de la herramienta están desarrollados de manera correcta.

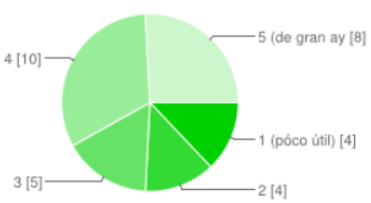


Esta respuesta muestra dos aspectos interesantes. El primero es que un 42% considera BoloTweet 2.0 una buena herramienta como complemento al estudio. El segundo aspecto, muestra quizás una carencia en cuanto a la formación y la información sobre la herramienta y sobre su objetivo.

Es importante para próximos usos de la herramienta, que antes de su utilización, se forme a los alumnos en el uso de la misma, y se les intente transmitir el objetivo que se quiere lograr con la utilización de *BoloTweet*, para no provocar usos de la misma si una intención definida.

Por otro lado, es positivo el reducido número de personas que la valoran negativamente.

¿Consideras útil tener una nota asociada a cada anotación?

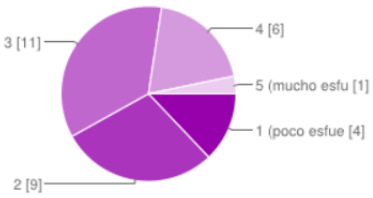


1 (póco útil)	**4**	13%
2	**4**	13%
3	**5**	16%
4	**10**	32%
5 (de gran ayuda)	**8**	26%

Esta respuesta muestra la aceptación de uno de los objetivos diferenciadores del resto de herramientas de micro-blogueo. Un 58% opina positivamente acerca del *feedback* proporcionado por las puntuaciones asociadas a las anotaciones.

Para próximas mejoras, será interesante proporcionar algún tipo de mecanismo que añada con un mínimo esfuerzo un breve comentario acerca de la anotación, de manera que los alumnos puedan obtener información acerca de sus *tweets*, más allá de una valoración numérica.

Valora el esfuerzo de realizar anotaciones cada clase

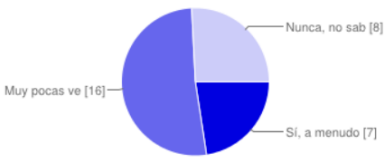


1 (poco esfuerzo)	**4**	13%
2	**9**	29%
3	**11**	35%
4	**6**	19%
5 (mucho esfuerzo)	**1**	3%

A pesar de que esta respuesta varía en gran medida dependiendo de la asignatura, es positivo que incluso con poca experiencia de uso, no consideren una tarea de mucho esfuerzo la síntesis y la generación de la anotación. Uno de los objetivos principales de BoloTweet 2.0 es simplificar y reducir el esfuerzo tanto para profesores como para alumnos, por lo que es una respuesta muy positiva.

No obstante, se espera que el esfuerzo sea cada vez menor a medida que se utilice con más continuidad.

¿Has utilizado en alguna ocasión la opción "Apuntes"?

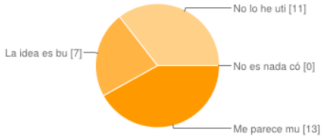


Sí, a menudo	**7**	23%
Muy pocas veces	**16**	52%
Nunca, no sabía ni que existía	**8**	26%

En esta respuesta se puede observar de nuevo una falta de información y de experiencia sobre la herramienta. Es importante, como se comentó anteriormente, que en próximos usos se les proporcione a los alumnos información suficiente como para que pueda utilizar todo el potencial de *BoloTweet*.

No obstante, los *Apuntes* no han estado disponibles todo el tiempo de uso de la herramienta, por lo que los resultados obtenidos son mejores de lo esperado.

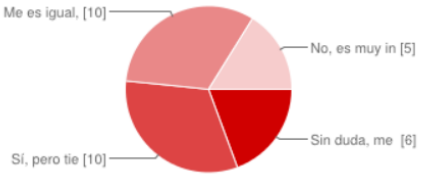
¿Qué opinas acerca del concepto de "Tareas"?



Me parece muy cómodo, ahorrándome tener que pensar los hahtags y la selección de grupo	13	42%
La idea es buena, pero al estar separado del grupo me ha provocado confusiones	7	23%
No lo he utilizado, prefiero enviar los mensajes a mano, escribiendo los hashtags y seleccionando el grupo	11	35%
No es nada cómodo	0	0%

Otro de los puntos diferenciadores e innovadores de *BoloTweet* revelan datos muy prometedores. El sistema de "Tareas" ha sido implementado en el último mes de uso, y únicamente ha sido utilizado por una asignatura, por lo que ha habido un gran porcentaje de estudiantes que no lo ha podido utilizar. Aun así, los resultados son muy positivos, teniendo que valorar en trabajos futuros la reubicación del mecanismo, pero apostando y continuando en esta misma línea.

¿Te gustaría utilizarlo en más asignaturas?



Sin duda, me ha sido de mucha ayuda para seguir la asignatura	6	19%
Sí, pero tiene que cambiar bastantes aspectos	10	32%
Me es igual, no le he prestado mucha atención	10	32%
No, es muy incómodo y no me ha ayudado	5	16%

Sin duda esta respuesta es indicadora del tiempo necesario de implantación de la herramienta. Hay muy buenas opiniones, y otras que lógicamente revelan la falta de uso, y el gran trabajo que falta por realizar.

Como última respuesta, los encuestados tenían la posibilidad de escribir opiniones, propuestas, o críticas hacia la herramienta:

Escribe opiniones o sugerencias que no hayas podido expresar anteriormente

Me parece una manera comoda de realizar feedbak entre alumnos . Es una buena herramienta para sintetizar las ideas y los conceptos en las asignaturas con alta carga teórica. Fuerza a los alumnos que asisten a clase periódicamente a "tomar apuntes" y eso es algo positivo. Me parece un buen proyecto y espero que su continuidad sea respaldada por la administración ya que aporta valor y ayuda a los alumnos. Quizá sea porque estoy acostumbrado a estudiar las cosas de pdf y libros, a mi no me ha sido muy útil. Creo que es una herramienta innecesaria, aunque la idea, lo que es el agrupamiento de ideas alrededor de un tema me parezca muy acertada. Quizá debería incorporarse a twitter, la necesidad de crear una nueva cuenta para usar la app, es aburrida Me parece una buena idea, ya que al final se supone que deberían estar sintetizadas las ideas importantes de una clase. Pero el hecho de que esté tan limitada la longitud del mensaje hace que ideas que no sean muy intuitivas simplemente nombrándolas, o explicaciones más largas de lo permitido, queden fuera, o muy sesgadas. Si esa longitud se aumentase lo suficiente como para que dejasen de ser frases sueltas, aunque se conservase alguna limitación para que no se convirtiese en un rinconDelVago, sería muy positivo. Si no vas a clase un día es muy últil para enterarte que han hecho o que han explicado. El mayor problema a la hora de generar apuntes es que el profesor ha puntuado con puntuaciones altas algunos comentarios que no tienen que ver con la asignatura, y el contexto de los demás no queda claro al tener separados tweets y fechas o deberían ir con el tema del que se habla en el tweet La herramienta apunta muy bien, pero le falta ese toce que diseño y usabilidad que la haría aceptable en el mercado.

5.4 Conclusiones

Como conclusiones, destacar como algo muy positivo el pequeño número de respuestas negativas. Esto abre un camino esperanzador, ya que no se rechaza la herramienta, si no que se plantean cambios, y sobre todo se refleja una falta de experiencia y de información.

En cuanto a los comentarios, se refleja por parte de varios alumnos el problema de la limitación de caracteres, ya que no es posible sintetizar ideas complejas de una manera correcta. Por lo general, son comentarios positivos, y sobre todo denotan que se han logrado algunos de los objetivos planteados al comienzo del proyecto.

6 Conclusiones

El propósito principal del proyecto era profundizar en el estudio de herramientas de micro-*blogging* como ayuda a la docencia. De manera secundaria, también se proponía personalizar, y modificar el sistema para añadir nuevas funcionalidades y mejoras, y por último, evaluar la utilidad de todo este entorno, y de la metodología de trabajo propuesta.

BoloTweet 2.0 ha sido probado cerca de 4 meses por 4 asignaturas y más de 100 usuarios. Las asignaturas pertenecían a cursos diferentes, por lo que también variaban la temática y los alumnos. El trabajo se ha hecho con un mínimo perjuicio para los estudiantes en términos de pérdida del servicio (por ejemplo durante actualizaciones del sistema) y del tiempo de los estudiantes y profesores, ya que se empleaba fundamentalmente el tiempo de clase.

No se ha hecho un análisis riguroso sobre la calidad de las anotaciones. La apreciación inicial es que todas las anotaciones expresan ideas muy representativas de las clases, lo que indicaría un alto grado de asimilación de contenidos por parte de los alumnos, y un buen ejercicio de síntesis. No obstante, el análisis de las anotaciones es un aspecto que no ha sido suficientemente explotado. El número de *tweets* por clase podría ayudar al profesor a conocer el ritmo de la clase. Un número bajo de tweets, puede indicar una clase demasiado compleja, o dificultad para comprender los contenidos impartidos. Sí que se puede destacar el uso correcto del sistema por parte de los alumnos. No ha habido casos de un abuso de la herramienta, limitándose en todo momento a utilizarla de manera únicamente educativa.

Como aspectos negativos, se han observado problemas y errores en la redacción de los mensajes, como etiquetas mal formadas o anotaciones sin referencias a ninguna asignatura. En ocasiones, esta mala redacción ha sido provocada por la limitación de caracteres, algo criticado por los alumnos. Este problema, además, se trata de solventar utilizando el mecanismo de Tareas incorporado en *BoloTweet 2.0* el cual, por falta de tiempo, no ha podido ser validado apropiadamente.

Otro aspecto en contra es la necesidad de dispositivos a través de los cuales realizar las anotaciones. Muchos de los alumnos disponen de portátiles, o dispositivos móviles con conexión a internet, pero hay otros que no.

Por lo tanto, hay que concluir que:

- **Hay un uso correcto del sistema por parte de los alumnos**. Una vez explicada la metodología de trabajo propuesta, los alumnos no incurren en ningún uso negativo del sistema.
- **Los alumnos se esfuerzan por sintetizar una idea**. Prácticamente todas las anotaciones generadas en el sistema muestran una idea relevante sobre una clase. Esto indica una buena asimilación de los conocimientos impartidos, y un buen ejercicio de síntesis.
- **Los estudiantes colaboran entre ellos**. Se han observado bastantes síntomas que indican que los alumnos utilizan el sistema más allá de comentar su idea. Colaboran, responden y ayudan a otros compañeros, incluso llegando a corregir si una idea no ha sido correctamente formulada, siguiendo un enfoque *student-teach-other-students* [7].
- **Creación de diferentes puntos de vista acerca de una clase**. Las anotaciones generadas sobre las clases contienen diferentes ideas, con pocas repeticiones. Esta disparidad beneficia aportando diferentes perspectivas acerca de una misma clase, y ofreciéndolos al resto de compañeros, lo que les permite obtener una visión más amplia del tema impartido.

- **Material de evaluación docente**. Al finalizar el curso, el profesor tiene de manera sencilla y rápida un mecanismo de evaluación individual, que puede emplear para valorar el progreso del alumno.
- **El esfuerzo de evaluar textos de esa longitud es mínimo**. A pesar de que esto es subjetivo, se ha apreciado por parte de los profesores una buena facilidad de evaluación, pudiendo incluso utilizar los 10 minutos de cada clase para corregir los del día anterior. No obstante, se proponen varias ideas como trabajo futuro, para permitir reducir este esfuerzo aún más.

Varios problemas y cuestiones están aún abiertas. No se ha podido probar en profundidad el impacto del nuevo mecanismo de "Tareas", ni de la opción de "Apuntes". En cuanto a la satisfacción de los objetivos, en la introducción se marcaban un objetivo metodológico y otro tecnológico:

Objetivo metodológico. Se ha ampliado la metodología de trabajo inicial. Algunas ideas estaban ya presentes en *Bolotweet 1.0*. Se han recogido y ampliado con las nuevas contribuciones de *Bolotweet 2.0.* Concretamente, se ha introducido el uso de *tareas* y *apuntes*, y se ha mejorado el mecanismo de puntuaciones asociado a *tweets*. Otras funciones previas se han seguido recogiendo, aunque adaptadas a la nueva versión de Status.Net que ha sido integrada en *Bolotweet 2.0.*

Objetivo tecnológico. Se ha creado herramientas de soporte adicionales que trataban defectos observados en versiones anteriores:

- En cuanto al problema de privacidad, se ha implementado un mecanismo completamente diferente de listado de puntuaciones, en los que un usuario solamente puede ver los grupos de los cuáles es miembro. Además, las puntuaciones únicamente están visibles para los profesores.
- Para resolver el problema de Administración revelado en la versión anterior, se han desarrollado un total de 8 herramientas que permiten al administrador realizar todas las tareas necesarias para la gestión de BoloTweet de manera automática. Además, se ha modificado el comportamiento interno de algunas partes del *core* de *StatusNet* para que estas herramientas funcionen sin suponer esfuerzo alguno.
- Comunicación con los miembros: Para solucionar esto, se han desarrollado 2 scripts diferentes. El primero es un script que se utiliza para enviar un correo de bienvenida a todos los usuarios en el momento de su registro, y poder informarles así del sistema y de sus funcionalidades. El segundo script, permite comunicarse con los miembros de manera individual, por asignaturas, o bien de manera global, y permite enviarles un correo electrónico personalizado con la información que se desee.
- Múltiples profesores y puntuaciones: Se ha modificado el funcionamiento interno de *Grades* por completo para soportar profesores y puntuaciones múltiples. Ahora, si hay más de un profesor vinculado a una asignatura, cualquiera de ellos puede puntuar el mismo *tweet*, generando una puntuación resultante con la media de ambas.
- Aprovechamiento de los *tweets*: Se ha desarrollado completamente desde cero un plugin que permite generar apuntes aprovechando las ideas de los alumnos. Estos apuntes se generan por grupos, y pueden producirse de manera automática, o personalizada, en la que el usuario puede seleccionar un determinado *hashtag*, usuario, o puntuación. El plugin genera un fichero resultante PDF con las ideas seleccionadas, así como una sección "autores" en la que se puede acudir en cualquier momento para conocer el autor de un determinado tweet.
- Profesores con contexto: Para solucionar esto se ha modificado la estructura interna de Grades. Ahora los profesores tienen asignaturas vinculadas, y solamente pueden puntuar mensajes dirigidos a esas asignaturas.

- Generación de anotaciones. Se ha desarrollado un mecanismo de apoyo a la metodología de trabajo propuesta a través del plugin Task. En este caso, los profesores pueden crear tareas, que no son más que un sistema en el que profesores pueden llevar un control del transcurso de la asignatura y de las clases, y los alumnos pasan a tener parcialmente automatizado el proceso de generación de anotaciones.
- De forma adicional, se ha conseguido enlazar de alguna manera *BoloTweet* con el CV, exportando las calificaciones del sistema en un formato comprensible por el CV.

Todas estas mejoras han sido valoradas por los estudiantes, predominando la impresión de que han conseguido ofrecer un sistema estable, y fácil de manejar. Las encuestas indican resultados positivos en el uso del sistema, si bien hay espacio de mejora todavía. Entre las mejoras que se prevén para un futuro, se encuentran:

- Valorar la integración de "Tareas" dentro de cada grupo.
- Introducir personalización de apuntes múltiple.
- Uso del sistema a gran escala.
- Generación de algún tipo de herramienta para recopilar estadísticas de uso.
- Mejora de la interfaz.
- Estudiar la posibilidad de migrar a *ShareTronix.*

Destacar por último la aceptación de *BoloTweet 2.0* por parte de la Universidad Complutense de Madrid como **Proyecto de Innovación y Mejora de la Calidad Docente** (PIMCD). Este hecho aporta un reconocimiento real como proyecto al trabajo realizado, además de proporcionar recursos para continuar estudiando el impacto de herramientas de micro-*blogging* en el ámbito docente, en este caso, a través de ***BoloTweet 2.0***.

7 Conclusions

The main purpose of the project was the study of micro-blogging tools as a teaching support tool. It also proposed customizing, and modifying the system to add new features and improvements. Finally, it evaluated the usefulness of the whole environment, and the methodology proposed from the perspective of students.

BoloTweet 2.0 has been running about 4 months. It has been used for 4 subjects and more than 100 users. The subjects belonged to different courses. The Bolotweet service was provided satisfactorily, with a minimal loss of service (eg for system updates).

The quality of annotations has not been properly studied, but an initial assessment is that all entries expressed ideas highly connected to the lectures, indicating a high level of assimilation of content by students, and a good synthesis practice. Annotations could be more extensively analyzed. For instance, the number of tweets per class could help the teacher to know the rhythm of the class. A small number of tweets could indicate a too complex, or difficult to understand, lecture. We can highlight the correct use of the system by students. There have been no identified cases of an abuse of the tool.

On the negative side, there have been problems and errors in the typing of the messages, as malformed tags or annotations without reference to any subject. Sometimes, these typos have been caused by the character limit, something criticized by the students. This problem is solved in part by using the built-in mechanism Task. However, its deployment was made late in the course, and could only be partially validated.

Another aspect is the need of a device through which the user can make annotations. Many of the students have laptops, or mobile internet devices, but others don't. For the later ones, students were allowed to complete the annotation at home.

Therefore, it can be concluded that:

- **There is a proper use of the system by students**. Having explained the proposed methodology, students use the system properly.
- **Students struggle to synthesize an idea.** Practically all system generated annotations show a relevant idea of the lecture. This indicates a good assimilation of knowledge, and a good synthesis practice.
- **Students collaborate among them.** It has been observed symptoms that many students use the system out of classroom. They collaborate by answering and helping other students, even pointing out mistakes if an idea has not been properly formulated. Though this tendency was small, it could be a step towards the student-teach-other-students approach [7].
- **Sharing different views about a lecture**. The generated annotations during lectures are mainly different, with little repetition of ideas. This allows students to know what other colleagues learnt and share their experience. To the teacher, it means an advance of what the class is understanding and if it progresses as expected.
- **Material teacher evaluation**. After completing the course, the teacher has a simple and quick individual evaluation mechanism, which can be used to evaluate student progress.
- **The effort to evaluate text of that length is minimal**. Although this is subjective, it has been remarked by teachers the easiness of the evaluation system. It fits the 10 minutes of each lecture to correct the previous day, but it requires a high discipline and a not so high number of students. Several ideas are proposed as future work to further reduce this effort.

Several problems and issues are still open. For instance, it has not been possible to test in depth the impact of the new mechanism "Tasks" or "Notes" options. Regarding satisfaction objectives, in the introduction, one methodological and technological objective were marked:

Methodological objective. It has expanded the initial working methodology. Some ideas that were already present in *Bolotweet 1.0* have been collected and expanded with new contributions. In particular, it has been introduced the use of tasks and notes, and an improved scores mechanism associated with tweets. Other previous functions have been adapted to the new version of Status.Net that has been integrated into Bolotweet 2.0.

Technological objective. It has created additional support tools dealing with observed defects in previous versions:

- On the issue of privacy, it has been implemented a completely different mechanism of listing of scores. Now, a user can only see the groups of which it is a member. In addition, scores are only visible to teachers.
- To solve the administration problems revealed in the previous version, it has been developed a total of 8 tools that allow the administrator to perform all necessary management tasks automatically. In addition, the internal behavior of some parts of the core *StatusNet* have been modified to support effortlessly some frequent admin tasks.
- Communication with members. Two different scripts have created to deal with this. The first is a script that is used to send a welcome email to all users at the time of registration, and to inform them about the system and its functionality. The second script, to communicate with members individually, by subject, or globally. It allows to send a personalized email with the desired information.
- Multiple teachers and scores. It has been changed the inner workings of Grades completely to support teachers and multiple scores. Now, if there is more than one teacher linked to a subject, any of them can score the same tweet, generating a resulting score with a weighted average method.
- Use of tweets. A plugin that can generate notes according to students' ideas has been completely developed from scratch. These notes are generated by groups, and can be produced automatically, or personalized way in which the user can select a particular hashtag, user, or score. The plugin generates a resulting PDF file with selected ideas, as well as an "author" section that can come at any time to meet the author of a given tweet.
- Teachers with context: To fix this, it was necessary to change the internal structure of the Bolotweet 1.0 Grades plugin. Now teachers have related subjects, and can only be rated annotations to these subjects.
- Generation of annotations. A mechanism to support the methodology proposed by the Task plugin was developed. In this case, teachers can create tasks. Tasks allows teachers to keep track of the annotation progress during the course and lectures, and it allows students to start annotations with a template instead of a blank annotation.
- Additionally, it has been possible to link BoloTweet with the UCM's Virtual Campus by exporting scores system in an understandable format for CV.

All these improvements have been rated by students. The general opinion is that they have managed to provide a stable and easy to handle system. Surveys indicate positive results in the use

of the system, although there is margin for improvement yet. Among the improvements are expected in the future are:

- Integrating "tasks" within each group instead of keeping it as a separated tool
- Further customization of the notes feature
- Using the system on a larger scale
- Collecting usage statistics.
- Improved user interface.
- Consider the possibility of migrating the system to the ShareTronix platform.

To conclude, this work has been part of a Project Innovation and Improvement of Teaching Quality (PIMCD) which was recently accepted by the Universidad Complutense de Madrid. This recognition brings some resources to continue this line of work.

8 Apéndices

Trabajo de Fin de Grado en Ingeniería del Software

Curso 2013 - 2014

BOLOTWEET 2.0

A. Manual de Uso - Usuario

Autor: Álvaro Ortego Marcos

ÍNDICE

Introducción. Qué es BoloTweet

BoloTweet es un método de trabajo basado en micro-anotaciones (*tweets*). Está desarrollado sobre la plataforma ***Status.Net***.

La idea de *BoloTweet*, es servir como herramienta de estudio para los alumnos, dónde podrán ir anotando ideas y comentarios sobre las asignaturas cursadas. Por otro lado, *BoloTweet* también está enfocado a ser una herramienta de ayuda a la enseñanza, pudiendo controlar el ritmo de la clase, tanto a nivel general, como particular, además de ofrecer un sistema de calificación.

Escribir un tweet

Para escribir un *tweet*, se utiliza el espacio reservado para ello, situado al comienzo de la página.

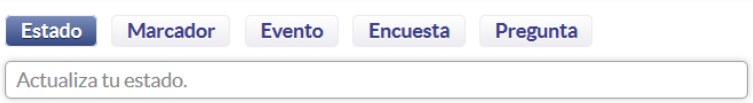


Ilustración 50 - Cuadro de anotaciones

- Estas micro-anotaciones, tienen una extensión máxima de **140** caracteres.

Elementos de un tweet

- **Tags** (*HashTags*): Los tags, o etiquetas, son palabras que marcan una información determinada. No hay límite de tags, siempre y cuando quepan en los 140 caracteres. Se considera tag todo lo que comience por '#'.

- **Grupos**: Los grupos en *BoloTweet* representan las asignaturas. A cada grupo pertenecen los estudiantes matriculados en las mismas. Para poder escribir a un grupo, y que el mensaje sea reconocido, es necesario formar parte de él.

  Se escribirá a un grupo a través del carácter **'!'**, seguido de un nombre de grupo existente:

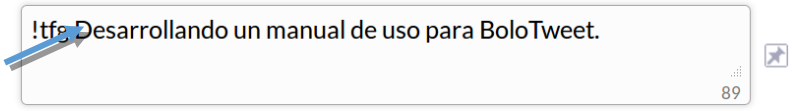


Ilustración 51 - Ejemplo de mecanismo de envío a grupo desde el tweet

  O bien seleccionando el grupo elegido al publicar el tweet.

Ilustración 52 - Ejemplo de selección de grupo mediante selector

Si se ha enviado correctamente, el tweet se mostrará de la siguiente manera:

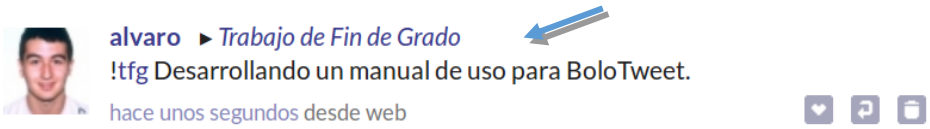


Ilustración 53 - Ejemplo de anotación enviada a un grupo concreto

Todos los grupos disponibles se pueden ver en el siguiente enlace:

https://grasia.fdi.ucm.es/bolotweet/groups

- *Los tags y los grupos permiten agrupar información. Al pulsar en cualquiera de ellos, se accederá a una página con todos los tweets relacionados.*

- **Enlaces**: Es posible añadir vínculos hacia páginas webs, y que estos sean reconocidos. Si son demasiado largos, el sistema automáticamente los acortará, mostrando un enlace menor.

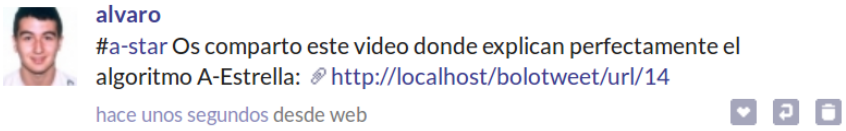


Ilustración 54 - Ejemplo de anotación con enlace

Es posible que algunas palabras con un punto entre medias, sean consideradas como un enlace, si el analizador de texto del sistema determina que lo es.

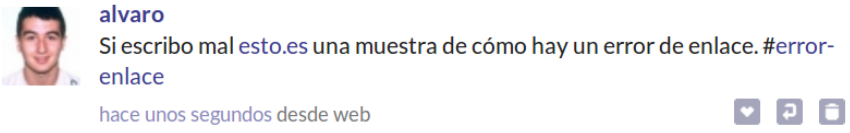


Ilustración 55 - Ejemplo de error de detección de enlace

Espacios de anotación

Las anotaciones publicadas, se pueden encontrar en diferentes lugares:

- **Línea temporal pública**. Esta línea es compartida por todos los usuarios. Aquí se muestran todos los tweets publicados, así como actividades y notificaciones. Se actualiza con gran frecuencia, por lo que es difícil de seguir.

Se accede a través del botón: 

- **Grupos.** En este espacio se muestran los grupos a los que perteneces. Si accedes a cualquiera de ellos, verás únicamente los mensajes y las anotaciones pertenecientes al grupo.

- **Línea de tiempo principal.** En este espacio se muestran tus propias anotaciones, actividades, y las de las personas que sigues.

Se accede a través del botón: 

- **Favoritos.** Este espacio contiene aquellas anotaciones marcadas como favoritas.

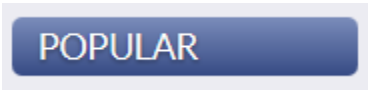
Se accede a través del botón: 

- **Perfil.** Este espacio muestra únicamente tus actividades, y tus anotaciones.

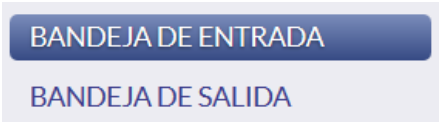
  Se accede a través del botón: 

- **Popular.** En este espacio se muestran los mensajes más populares del sitio, es decir, aquellos que más gustan a los usuarios.

  Se accede a través del botón: 

- **Mensajes.** En este apartado se muestran tus mensajes privados, tanto la bandeja de entrada, como de salida.

  Se accede a través del botón: 

- **Respuestas.** Este espacio contiene todas las respuestas recibidas a cualquiera de tus anotaciones.

  Se accede a través del botón: 

Acciones sobre Tweets

Además de escribir anotaciones, *BoloTweet* ofrece algunas acciones extra.

- **Responder**: Existe la posibilidad de responder un tweet ya publicado. Como es lógico, la respuesta a un tweet debe estar relacionada con el tema del mismo.

- **Compartir** (*retwittear*): Un *tweet* publicado puede ser compartido por el resto de usuarios. Al compartir un *tweet*, se está publicando de nuevo, y haciéndose visible para tus seguidores. Es una buena forma de dar popularidad a un *tweet* interesante.

- **Favorito**: Si un tweet es de tu agrado, puedes indicar que te gusta.

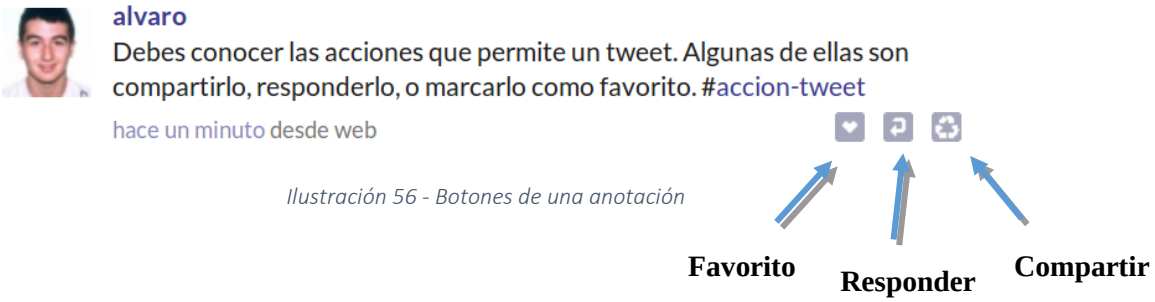


Ilustración 56 - Botones de una anotación

Más opciones de Bolotweet

Este apartado estará en constante evolución, a medida que se vayan añadiendo nuevas funcionalidades.

- **Preguntas**: Existe la posibilidad generar una pregunta como tal. Esta pregunta, irá dirigida a quien tú decidas, y podrá ser contestada por cualquier persona.

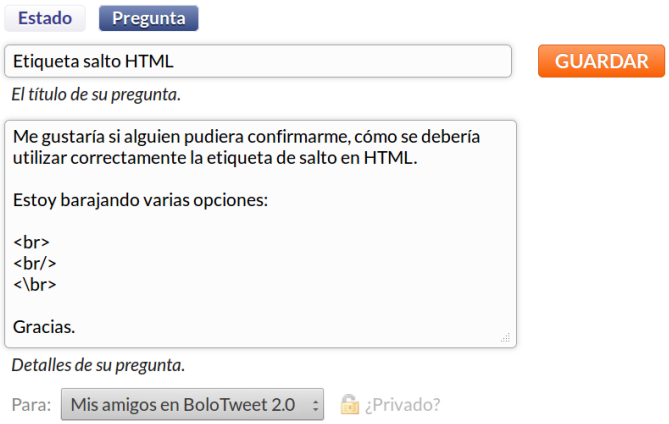


Ilustración 57 - Formulario de creación de preguntas

Una vez desarrollada la pregunta de la mejor forma posible, la pregunta quedará abierta a respuestas.

Cuando se reciba una respuesta de nuestro agrado, se debe marcar como ***Mejor***, y se cerrará automáticamente la pregunta.

Si por alguna circunstancia no se obtuvieran respuestas, es posible cerrar manualmente la pregunta.

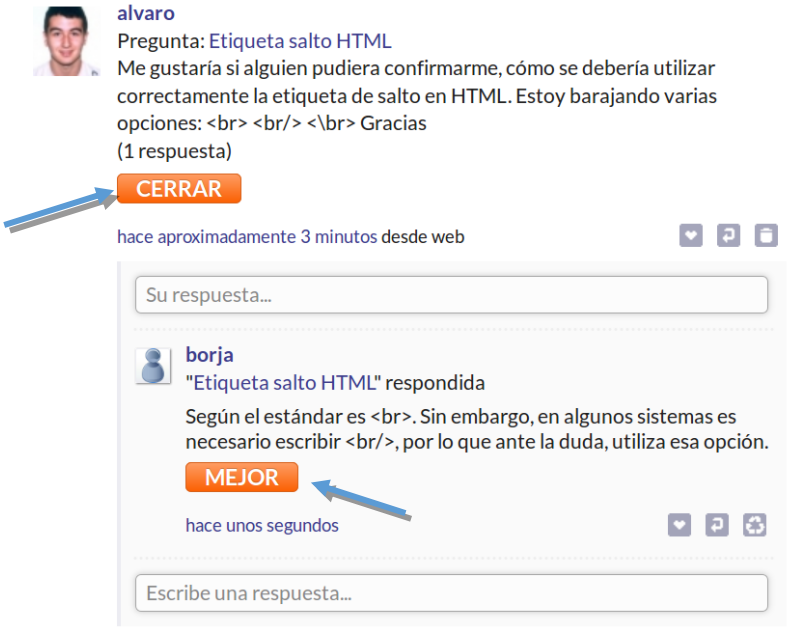


Ilustración 58 - Ejemplo de mecanismo de pregunta

- **Eventos**

Los eventos solamente pueden ser creados por profesores. Cuando un profesor crea un evento, es posible indicar la asistencia al mismo.

Para ello, desde el grupo en el que ha sido publicado, o bien desde el *Timeline* Principal, se puede pulsar la opción que se crea conveniente.

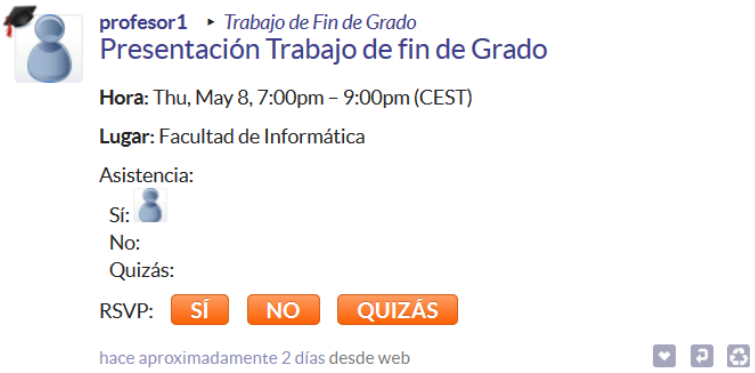


Ilustración 59 - Ejemplo de evento

- **Encuestas**:

La creación de encuestas, al igual que los eventos, está restringida únicamente a profesores. Las encuestas aparecerán en el grupo relacionado, o bien en el *Timeline* Principal.

Para responderla, tan sólo se deberá escoger la opción que se considere oportuna, y pulsar el botón enviar:

*** La opción enviada no se puede modificar.*

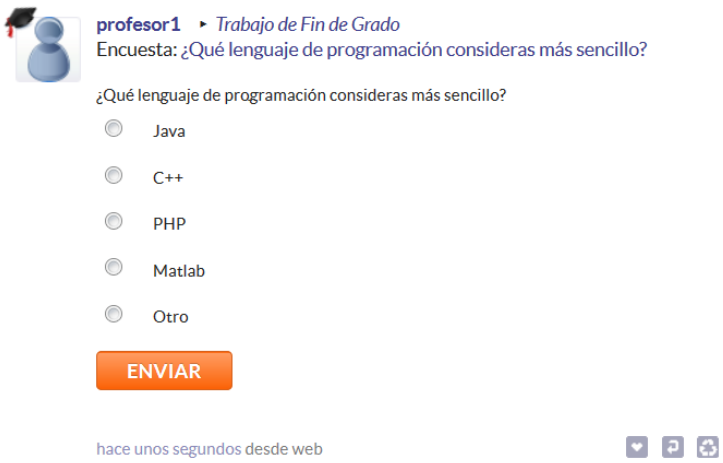


Ilustración 60 - Ejemplo de encuesta

- **Apuntes**: Este herramienta permite obtener un conjunto de *tweets* para un grupo determinado.

Estas *notas* estarán formadas por las ideas publicadas por los propios usuarios, y a pesar de servir de ayuda al estudio, en ningún caso sustituirá a los apuntes impartidos por el profesor.

Se accede a través del botón: 

Existen dos tipos de apuntes, personalizados, y automáticos.

Los apuntes automáticos generarán un PDF con la recopilación de los mejores tweets de un grupo determinado.

La opción personalizada, permite escoger manualmente apuntes para un *tag* determinado, un alumno en concreto, o una nota específica.

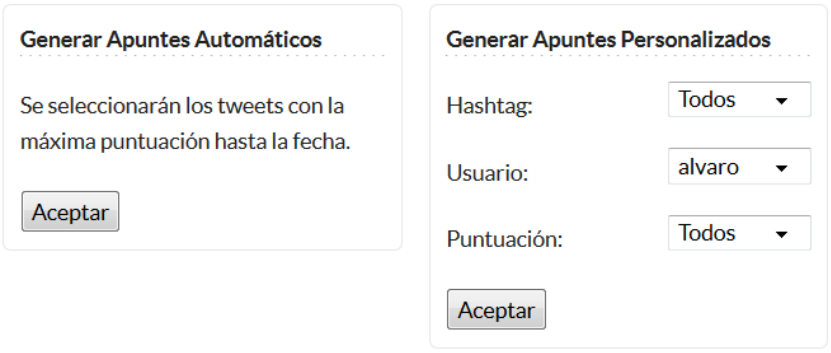


Ilustración 61 - Formulario de selección de apuntes

- **<u>Búsqueda Usuarios</u>**: Si deseas buscar un usuario en concreto, se puede hacer a través de la opción *Directorio*. Podrás navegar entre los usuarios a través de sus iniciales, o bien utilizando el buscador.

  Se accede a través del botón: 

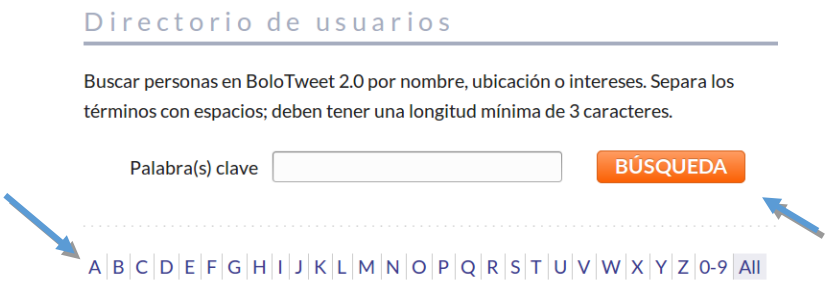


Ilustración 62 - Directorio de usuarios

- **<u>Grade Reports</u>**: En cada asignatura, las *micro-anotaciones* que se publiquen podrán ser puntuadas por los profesores. Con *Grade Reports*, se podrá visualizar un listado para cada grupo al que perteneces, con un ranking de alumnos, de acuerdo a la puntuación total de sus *tweets*.

  Se accede a través del enlace: 

Metodología de trabajo

Con todas las herramientas proporcionadas por *Bolotweet*, y para mantener un cierto orden, se deberá seguir una metodología de trabajo.

- **Resumen del día**

Los *tweets* de una determinada asignatura, deberán ir dirigidos a ese grupo en concreto, como se explicó en la sección correspondiente, además de añadir un *tag* (**#yyyy-mm-dd**) con la fecha.

- *Ejemplo: El estudiante realiza un resumen de la lección del **14 de Marzo de 2014**, para la asignatura Trabajo fin de grado, que se corresponde con el grupo "**tfg**".*

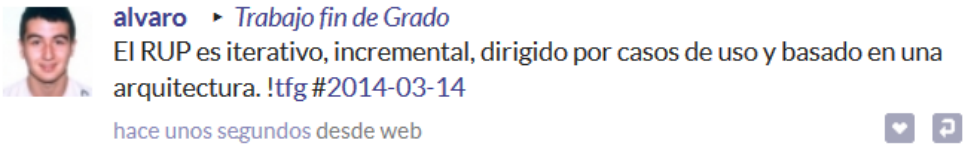


Ilustración 63 - Ejemplo de resumen diario

Tareas

Para facilitar la creación del "resumen diario", se ha desarrollado un nuevo concepto denominado "Tareas". Esta herramienta es complementaria al mecanismo anterior, pudiendo utilizar el que resulte más cómodo, e incluso pudiendo escribir más *tweets* que tareas existentes.

Las tareas automatizan la metodología de trabajo, de manera que el usuario tan sólo debe escribir su anotación, olvidándose de etiquetas con la fecha, con el grupo…

Cuando un profesor cree una Tarea, aparecerá un aviso junto al menú "Tareas", situado en la barra lateral izquierda:

Ilustración 64 - Ejemplo de notificación de tareas pendientes

Al acceder, se mostrará una lista con todas las tareas pendientes.

Ilustración 65 - Ejemplo de lista de tareas pendientes

Será tan sencillo como pulsar "Completar", o bien "Rechazar", si no se quiere contestar, escribir la idea que se quiera, y pulsar enviar.

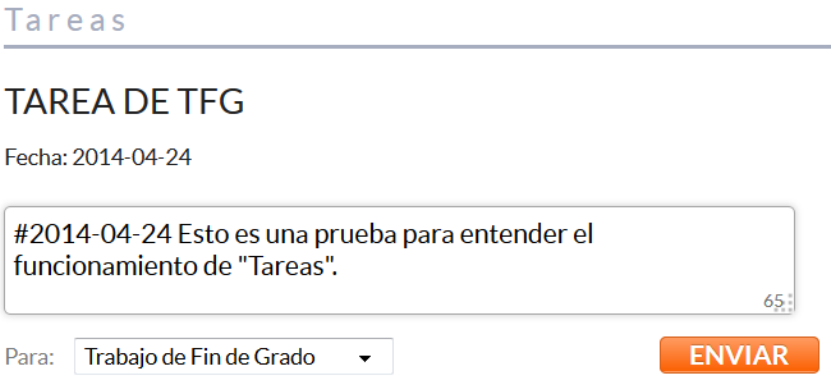


Ilustración 66 - Mecanismo de envío de tarea

Al pulsar "Completar", la etiqueta con la fecha se cargará automáticamente, y en el caso de que la tarea lleve una etiqueta extra relacionada, también se mostrará de forma automática.

Por otro lado, no será necesario escribir a qué grupo va destinado, ya que se enviará automáticamente al grupo que corresponda.

- **Crítica a la lección**

Es posible indicar una cierta crítica a una asignatura, o a una clase en concreto. Para que se considere, y sea tratada como tal, se deberá añadir el *tag* (***#crit***), además de cualquier otra etiqueta que ayude a la formulación de la crítica.

Es fundamental indicar qué es lo que no gusta, y por qué.

- *Ejemplo: La lección de "**tfg**" del **10 de Marzo de 2014** ha ido demasiado rápido.*

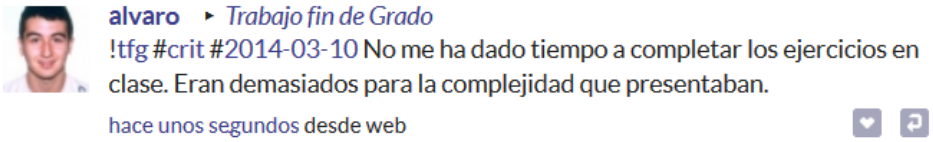


Ilustración 67 - Ejemplo de anotación de crítica 1

- *Ejemplo: El alumno del curso "**tfg**" echa en falta más ejemplos de un tema concreto.*

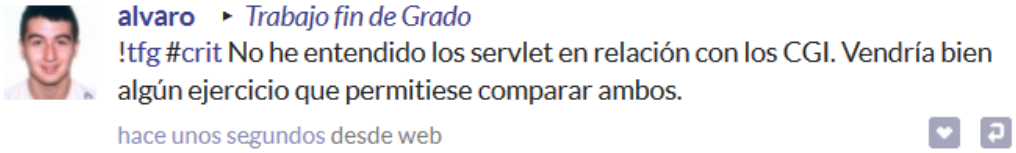


Ilustración 68 - Ejemplo de anotación de crítica 2

- **Preguntas**

Para la formulación de preguntas, se utilizará el mecanismo de preguntas explicado anteriormente, siempre dirigiéndolo a un grupo concreto, y con información precisa sobre la cuestión.

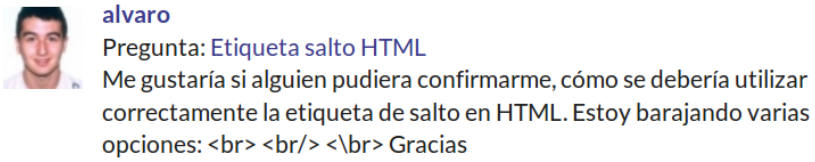


Ilustración 69 - Ejemplo de pregunta

Privacidad

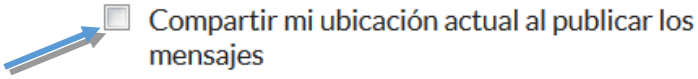
Todos los usuarios aceptan, por el hecho de registrarse, que los contenidos que pongan en el sistema son accesibles por otros usuarios registrados.

Estos contenidos están sujetos a la licencia ***Creative Commons***, pudiendo reutilizarse en otros contextos siempre que se haga una mención al autor original. Los términos generales de privacidad están accesibles desde el pie de página en todo el sistema.

Reconocimiento – NoComercial – CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Ilustración 70 - Licencia de uso de BoloTweet 2.0

Cada entrada en el sistema puede estar marcada con la ubicación geográfica del usuario. Para evitarlo, hay que desactivar la opción de geolocalización en **Ajustes**.



Por ser alumnos de la Universidad Complutense de Madrid, al utilizar *BoloTweet* se acepta, y se debe respetar por tanto, el código de conducta de la UCM.

http://pendientedemigracion.ucm.es/cont/descargas/documento28854.pdf

Trabajo de Fin de Grado en Ingeniería del Software

Curso 2013 - 2014

BOLOTWEET 2.0

B. Manual de Uso - Profesor

Autor: Álvaro Ortego Marcos

ÍNDICE

Introducción. Qué es BoloTweet

BoloTweet es un método de trabajo basado en micro-anotaciones (*tweets*). Está desarrollado sobre la plataforma ***Status.Net***.

La idea de *BoloTweet*, es servir como herramienta de estudio para los alumnos, dónde podrán ir anotando ideas y comentarios sobre las asignaturas cursadas. Por otro lado, *BoloTweet* también está enfocado a ser una herramienta de ayuda a la enseñanza, pudiendo controlar el ritmo de la clase, tanto a nivel general, como particular, además de ofrecer un sistema de calificación.

Solicitud Profesor y Grupo

Para solicitar ser profesor, y tener un grupo asignado, deberá enviarse un correo electrónico a la dirección indicada a continuación con los siguientes datos:

- Nombre: [Nombre completo del profesor]
- Nombre de usuario: [*Nick* escogido por el profesor]
- Email: [Correo electrónico del profesor]
- Grupo: [Creado/Nuevo]
- *Nick* del grupo: [*Nick* escogido para el grupo]

Correo electrónico: bolotweetSolicitud@gmail.com

Esta solicitud será validada por el administrador del sistema, y se procederá a la creación del usuario y del grupo.

Cuando este proceso haya finalizado, se le enviará un correo de vuelta con indicaciones.

El apartado de "Grupo" se utilizará para aquellos casos en los que una asignatura tenga más de un profesor, y alguno de ellos no esté registrado. De esta manera, se indicará que el grupo está creado, y con el resto de datos se creará el usuario, y se vinculará a la asignatura que corresponda.

Es importante conocer que el grupo se creará en modo "Privado", pudiendo cambiar esta configuración más adelante tal y como se explica en la sección *Modificar la Asignatura*.

Crear nueva asignatura

En el caso de estar ya registrado, y querer crear una nueva asignatura, se puede hacer manualmente desde el menú "Grupos", situado en el lateral izquierdo de la página, y posteriormente a través de la opción "Crear un grupo nuevo" (*Opción restringida únicamente a Profesores anteriormente creados*).

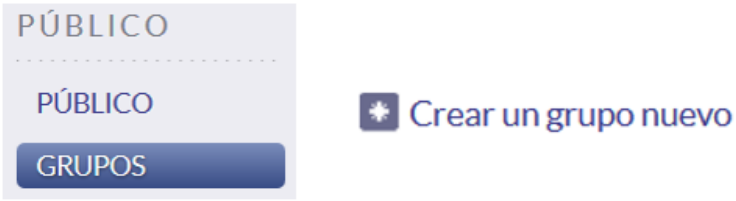


Ilustración 71 - Crear nuevo grupo desde la interfaz

Deberán rellenarse los campos que se consideren oportunos del formulario (*Imagen 1),* con las indicaciones de la siguiente sección.

Una vez creado, ese profesor automáticamente pasará a ser Profesor y Administrador del grupo, por lo que no es necesario enviar ningún correo electrónico adicional solicitando que se vincule el usuario con el nuevo grupo creado.

Solamente el Administrador del grupo podrá manualmente vincular más profesores a ese grupo. *(Véase sección "Agregar profesores a un grupo")*

Modificar la asignatura

Para modificar la información relativa a la asignatura, o si se ha creado el grupo a través de una solicitud por correo, puede hacerse a través del nombre del grupo en la columna de la izquierda, y a continuación pulsando en la opción "*Editar"*.

Ilustración 72 - Ejemplo de edición de asignatura

En el formulario que se muestra (*Ilustración 73*) se deberá ir completando la información que se desee relativa a la asignatura.

Nickname:	[Siglas o abreviatura escogida.	]
Nombre Completo:	[Nombre completo de la asignatura.	]
Página de Inicio:	[Página web asociada (blog, campus virtual…)	]
Descripción:	[Breve información sobre la asignatura.	]
Ubicación:	[Facultad, Escuela, Universidad.	]
Alias:	[Nombres adicionales con los que identificar al grupo.	]

Si se activa el campo **Privado**, cualquier usuario que desee unirse a la asignatura deberá ser aprobado por el administrador del grupo. Además, los mensajes serán ocultos a cualquier usuario que no pertenezca al mismo.

Un grupo configurado como **No privado** permitirá a cualquier usuario unirse al mismo sin confirmación alguna. Además, los mensajes podrán ser vistos por cualquier usuario de *BoloTweet*.

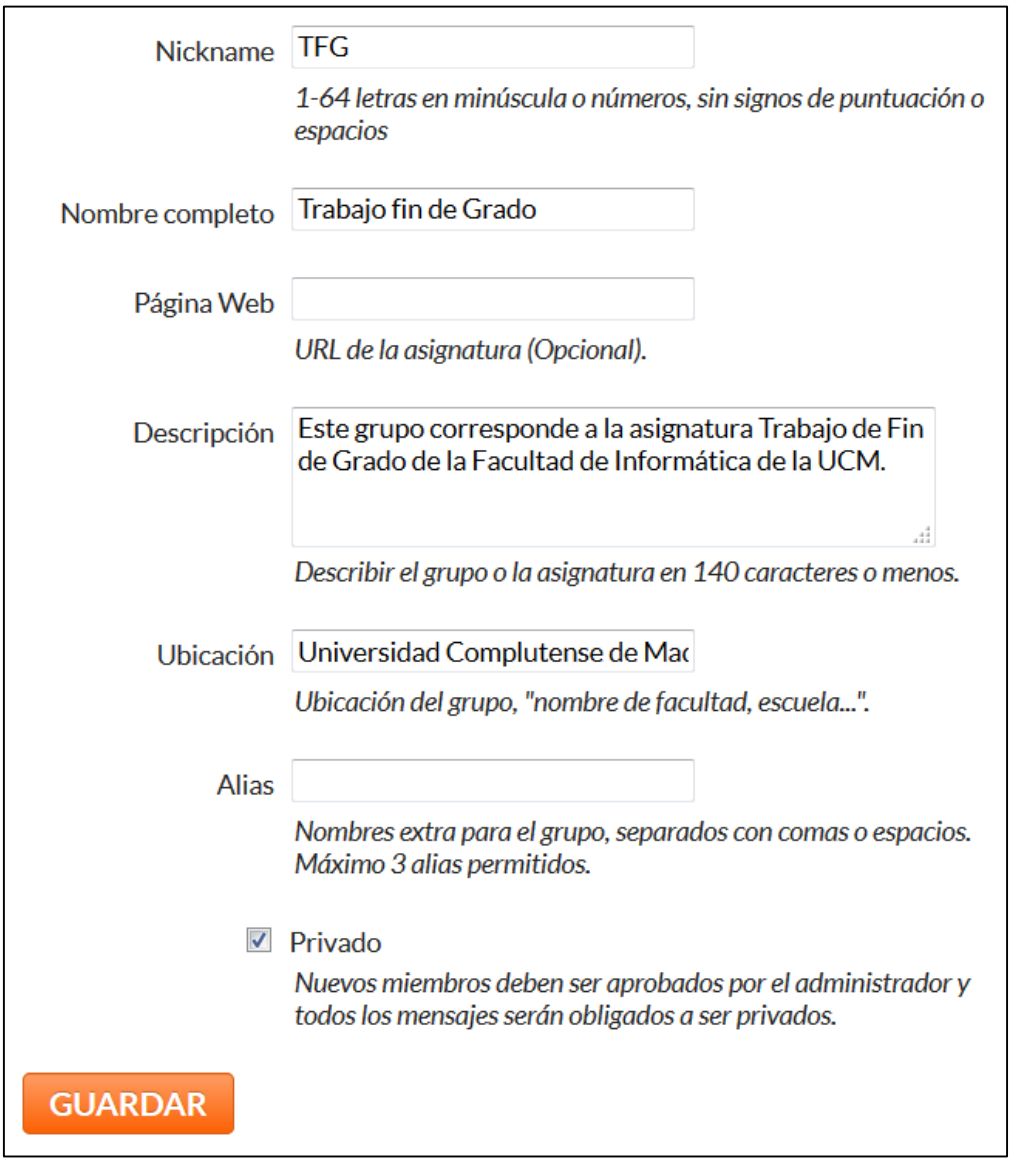


Ilustración 73 - Ejemplo de formulario de modificación de asignatura

Agregar profesores a un grupo

Si en una asignatura ya creada se desea agregar más de un profesor, puede hacerse a través de dos vías.

La primera y más sencilla es de manera gráfica. Para ello, el Administrador del grupo, deberá acudir a la sección "Miembros" del grupo en cuestión.

Ilustración 74 - Ejemplo de sección de miembros de grupo

Tras pulsar en el enlace, se mostrará la lista de usuarios pertenecientes al grupo, y una serie de opciones para cada uno.

Simplemente se deberá buscar el usuario que se desee, y pulsar en el botón "Hacer Grader".

Automáticamente se le asignará el rango de *grader* al usuario, y se le vinculará al grupo como profesor.

La otra vía, es solicitarlo a través de correo electrónico, indicando el nombre de usuario del profesor, y el *nick* del grupo al que se desea vincular. *(El usuario por tanto debe estar registrado, si no, véase sección "Solicitud Profesor y Grupo")*

Correo electrónico: bolotweetSolicitud@gmail.com

Cuando se haya vinculado, se recibirá un correo electrónico indicándolo.

Insertar Usuarios

Una vez creado el grupo y añadida la información que se desee, falta añadir los alumnos.

Para ello, el profesor tan sólo tendrá que generar un archivo Excel (**.xls, *.xlsx*) con la lista de los alumnos, y enviar un correo electrónico al administrador con el fichero.

El fichero debe respetar el siguiente formato:

- El nombre del fichero debe ser el *nickname* del grupo.
- La primera línea debe ser el nombre de la asignatura.
- La segunda línea debe contener el nombre de las columnas.
- El fichero debe contener 4 columnas con el siguiente orden:
  apellido1 -- apellido2 -- nombre – email
- Cada usuario estará en una nueva línea.

Ejemplo:

tfg.xls

1	Trabajo de Fin de Grado			
2	**Primer apellido**	**Segundo apellido**	**Nombre**	**E-mail Universidad**
3	ORTEGO	MARCOS	ÁLVARO	alvorteg@ucm.es
4	APELLIDO1	APELLIDO2	NOMBRE1	nombre@ucm.es

Ilustración 75 - Ejemplo de formato de fichero Excel

Tareas

Una de las novedades de *BoloTweet 2.0* es la aparición de "Tareas". Una tarea no es más que un mecanismo para facilitar al alumno el desarrollo de *tweets* relacionados con una asignatura.

Las micro-anotaciones publicadas por los alumnos, deben seguir una metodología de trabajo específica, y común a todas las asignaturas. (*Consultar en la sección "Metodología de Trabajo" del Manual de Usuario*).

Estas normas se ven automatizadas por el concepto de "Tarea", facilitando al alumno el uso del sistema.

Crear Tareas

Para crear una nueva Tarea, se deberá acudir al menú "*Tareas*" situado a la izquierda de la página.

Ilustración 76 - Opción tareas de la interfaz

Se desplegará una lista en la que se podrá iniciar una tarea para cada grupo vinculado al profesor.

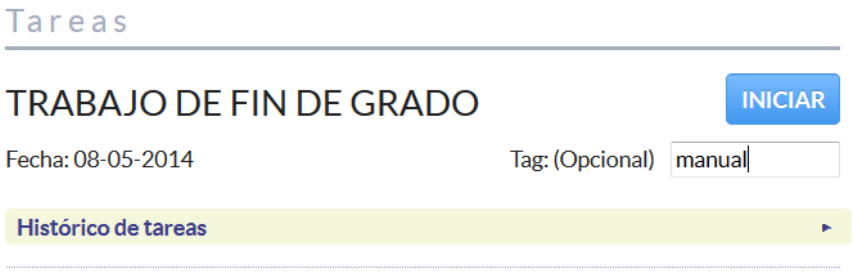


Ilustración 77 - Ejemplo de tarea para profesor

El procedimiento para iniciar la tarea es tan sencillo como pulsar el botón "*Iniciar*" y, opcionalmente, añadir una etiqueta (*tag)* relacionada con la tarea.

Si todo ha salido satisfactoriamente, en el Histórico de tareas aparecerá la nueva tarea creada, con información sobre la misma.

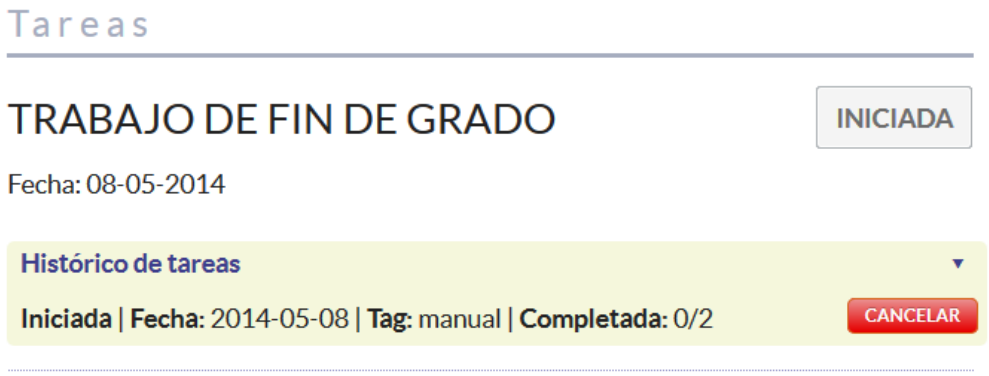


Ilustración 78 - Ejemplo de histórico de tareas

- El primer campo indica el estado de la tarea, (Cancelada/Iniciada).
- El segundo campo indica la fecha de creación de la tarea.
- El tercer campo muestra la etiqueta opcional vinculada con la tarea.
- El cuarto y último campo muestra cuántos de los alumnos la han completado.

Las tareas son diarias, por lo que cada profesor podrá iniciar para cada grupo como máximo una tarea al día.

Si en una asignatura participan más de un profesor, en este caso, cada uno de ellos podrá crear una tarea para una misma asignatura y en el mismo día.

Cancelar Tareas

Para cancelar una tarea previamente creada, es imprescindible que no haya sido contestada por ningún alumno.

Si esto se cumple, en el histórico de tareas aparecerá al lado de la tarea el botón para cancelar.

Una tarea cancelada automáticamente dejará de aparecer a los alumnos como “Pendiente”.

*** Es posible reabrir la tarea en cualquier momento desde el histórico pulsando Iniciar.*

Editar Tag de Tareas

En caso de querer modificar una etiqueta vinculada a una tarea, se ofrece un mecanismo para su actualización.

Sin embargo, es imprescindible realizarlo el mismo día de la creación de la tarea, y que esta no haya sido contestada por ningún alumno.

Para editar el *tag*, se deben seguir los siguientes pasos:

1. Cancelar la tarea desde el histórico de tareas
2. Colocar el nuevo *tag* en el espacio reservado
3. Pulsar el botón Iniciar (azul)

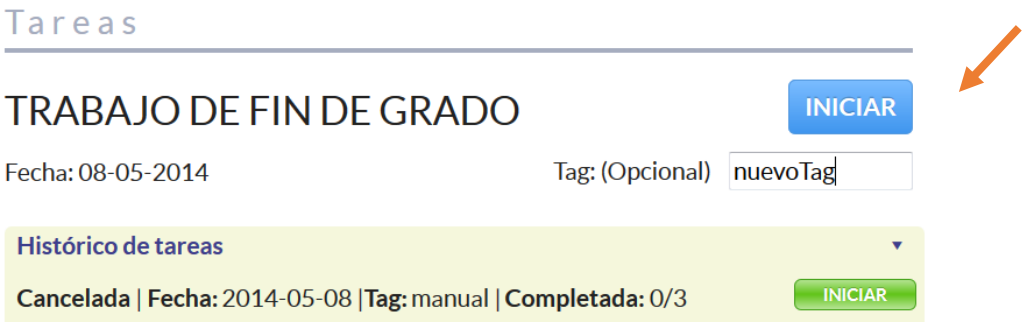


Ilustración 79 - Ejemplo de edición del tag asociado a una tarea

Tras realizar estos pasos, la tarea se reabrirá con el *tag* corregido, la tarea quedará iniciada y vinculada con la nueva etiqueta a todos sus alumnos.

Puntuar micro-anotaciones

Un profesor podrá puntuar cualquier *tweet* dirigido a un grupo del cual es profesor.

El proceso de calificación es extremadamente sencillo.

Cuando un *tweet* es susceptible de ser calificado por el usuario actual, automáticamente aparecerán unos botones para puntuarlo.

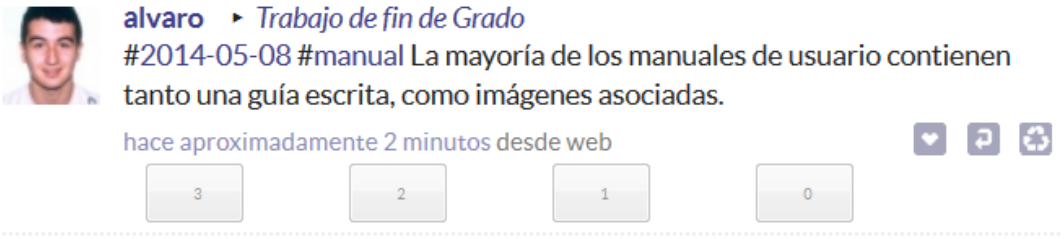


Ilustración 80 - Ejemplo de tweet con botones para calificar

Para calificar la *micro-anotación*, deberá pulsarse sobre cualquiera de ellos, teniendo en cuenta que el 3 es la máxima puntuación, y el 0 la menor.

Una vez puntuado, la próxima vez que se recargue la página aparecerá la calificación asociada.

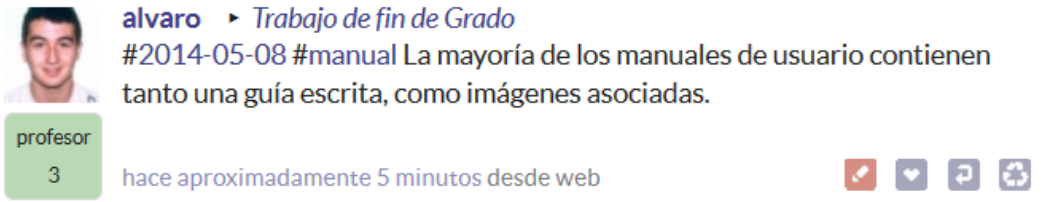


Ilustración 81 - Ejemplo de anotación calificada

Puntuaciones Múltiples

Otras de las novedades de la nueva versión es la aparición de calificaciones múltiples. Cuando una asignatura es impartida por más de un profesor, cada uno de ellos individualmente puede puntuar el mismo *tweet*, originando como calificación resultante una media aritmética de ambas.

En el caso de existir puntuación múltiple, el nombre que aparecerá en la calificación será "*Nota*", y se podrá ver el desglose de las mismas pulsando sobre el recuadro con la puntuación.

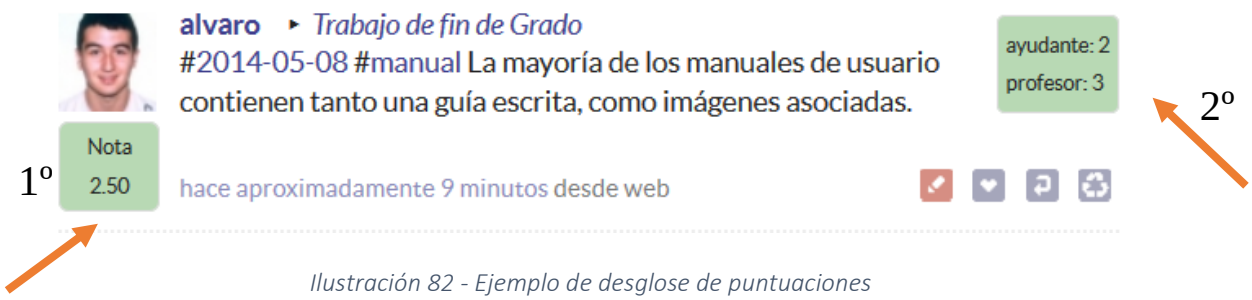


Ilustración 82 - Ejemplo de desglose de puntuaciones

Modificar puntuaciones

En el caso de puntuar un tweet de manera errónea, es posible modificar la puntuación de una manera sencilla. Para ello, tan sólo habrá que pulsar en el botón rojo en forma de lápiz.

Automáticamente se mostrarán de nuevo los botones para puntuar, y se podrá escoger la calificación correcta.

*** Sólo es posible modificar las puntuaciones realizadas por el propio usuario.*

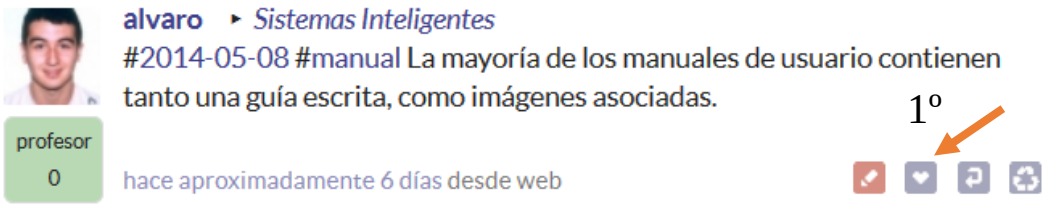


Ilustración 83 - Botón para modificar la calificación asociada a un tweet

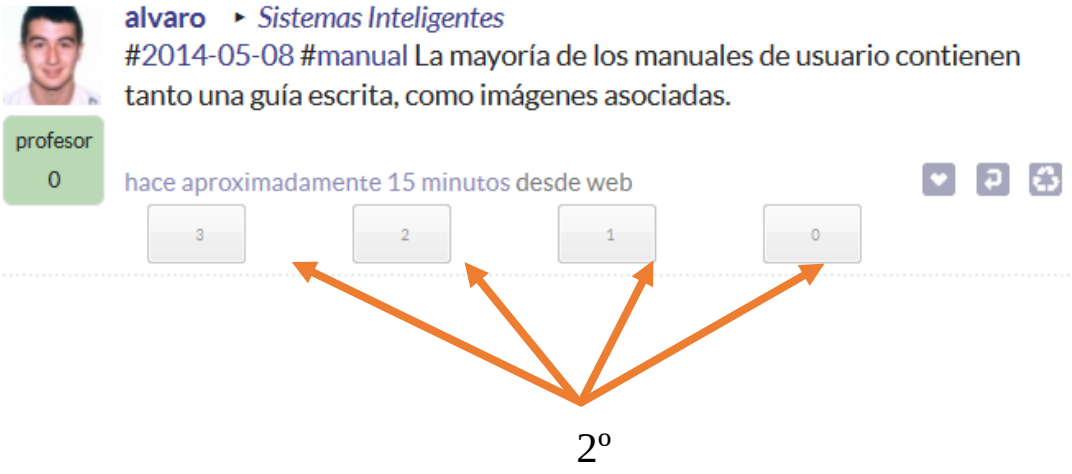


Ilustración 84- Ejemplo de modificación de una puntuación

Listado de Calificaciones

BoloTweet ofrece la posibilidad de mostrar un listado con las calificaciones asociadas a los alumnos para cada grupo al que pertenece

Este listado sólo muestra los grupos a los que pertenece el usuario actual.

Para acceder al listado, se debe pulsar el enlace situado al comienzo de la página "*Grade Reports*".

Ilustración 85 - Opción Grade Reports en la interfaz

Para cada grupo vinculado, se mostrará una lista con todos los alumnos, ordenados por puntuación total mayor.

Ilustración 86 - Ejemplo de listado de Grade Reports

Revisar la actividad de un usuario concreto

Si se desea revisar la actividad de un usuario determinado en un grupo concreto, por ejemplo para observar sus *tweets*, se puede hacer desde el propio *Grade Reports*.

Tan sólo habrá que pulsar en el nombre del alumno que se desee, y a continuación se cargará una página con los *tweets* de ese alumno en el grupo, así como unas pequeñas estadísticas acerca de la actividad del usuario.

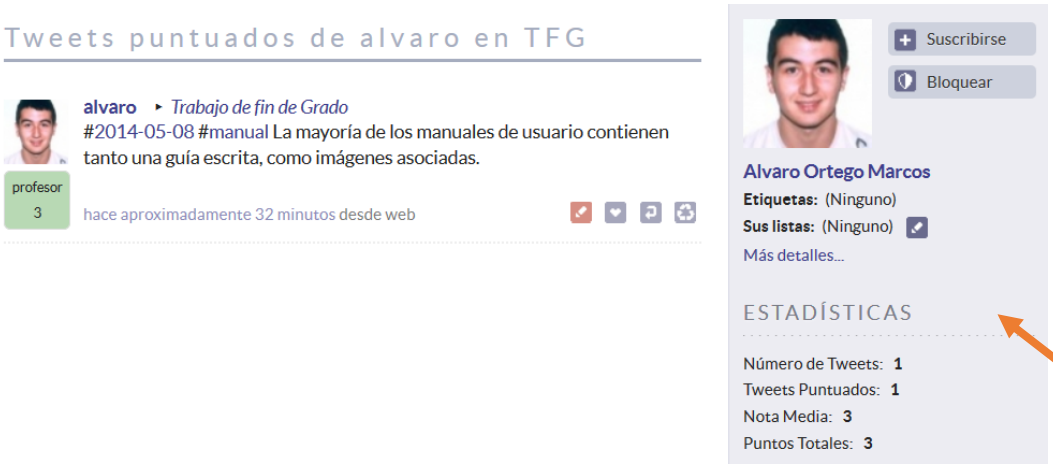


Ilustración 87 - Ejemplo de la actividad de un usuario en un grupo concreto

Exportar calificaciones

Otra de las opciones que se ofrecen, es la posibilidad de exportar la lista de calificaciones de un grupo determinado a un fichero CSV con el siguiente formato:

nickAlumno,puntuacionTotal

Para llevar esto a cabo, tan sólo se debe pulsar sobre el botón "Exportar CSV" (ver *Ilustración 86*), configurar las opciones de exportación, y pulsar "Generar CSV".

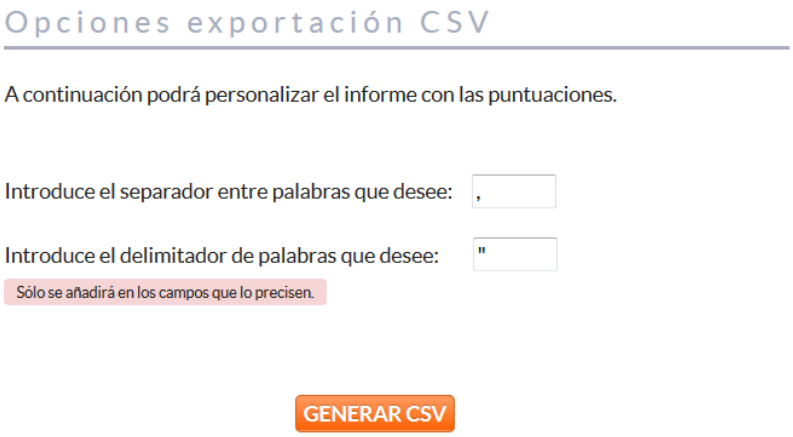


Ilustración 88 - Ejemplo de opciones de exportación de fichero CSV

Crear Eventos

Los profesores también podrán crear Eventos para cualquier de sus asignaturas, o para la comunidad *Bolotweet* entera.

Conferencias, salidas, viajes… cualquier evento de interés, puede ser creado a través de Bolotweet, y dirigido a un grupo específico.

A esta opción se accede a través del botón "*Evento"*, situado al comienzo de la página, y donde se deberá rellenar la información que se considere oportuna.

Ilustración 89 - Ejemplo de creación de evento

El evento le aparecerá a todos aquellos usuarios a los que se haya dirigido en el campo "*Para*", y serán estos quiénes podrán confirmar la asistencia al mismo.

Crear Encuestas

Otra de las opciones que se ofrecen a los profesores es la posibilidad de crear Encuestas con hasta 5 opciones para sus alumnos.

Las encuestas irán dirigidas al grupo que se especifique, y se podrán controlar los resultados siempre que se desee.

Para crearlas, se debe pulsar el botón "*Encuesta*", situado al comienzo de la página. Posteriormente habrá que rellenar la información que se considere oportuna, y se pulsará "*Guardar*".

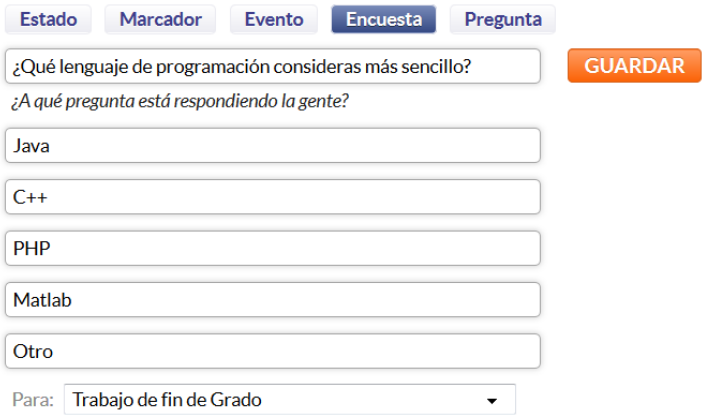


Ilustración 90 - Ejemplo de creación de encuesta

Resultados:

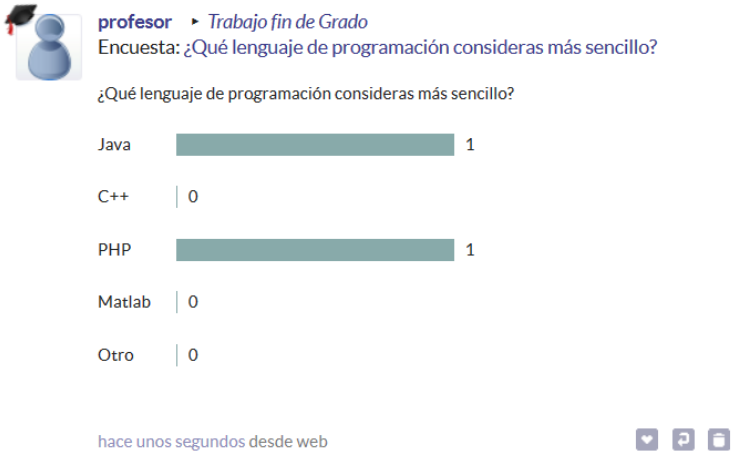


Ilustración 91 - Ejemplo de resultados de encuesta

Trabajo de Fin de Grado en Ingeniería del Software

Curso 2013 - 2014

BOLOTWEET 2.0

C. Manual de Uso - Administrador

Autor: Álvaro Ortego Marcos

ÍNDICE

Alta profesor y grupo

Cuando se quiere dar de alta una nueva asignatura, el profesor en cuestión deberá mandar un correo electrónico con la siguiente información:

- Nombre: [Nombre completo del profesor]
- Nombre de usuario: [*Nick* escogido por el profesor]
- Email: [Correo electrónico del profesor]
- Grupo: [Creado/Nuevo]
- *Nick* del grupo: [*Nick* escogido para el grupo]

Una vez recibidos los datos, hay que pasar a dar de alta todo lo necesario.

*** Para la administración de bolotweet, se utilizarán fundamentalmente Scripts a través de línea de comandos.*

Crear Usuario

registeruser.php [options]

-n --nickname	*Nick* escogido por el profesor
-w --password	Escribir cualquier contraseña
-f –fullname	Nombre completo del profesor
-e --email	Correo electrónico del profesor

El campo *password* debe ser rellenado con cualquier contraseña al azar, ya que inmediatamente después del registro se enviará un correo electrónico de bienvenida al usuario donde se le informará acerca de cómo cambiarla.

Tras ejecutar el script rellenando los campos con los datos recibidos, un mensaje indicará que el registro ha sido satisfactorio.

Ejecución:

```
alvaro@alvaro-linux:/var/www/bolotweet/scripts$ php registeruser.php -nalumno -f"Alumno Prueba Manual" -wpass -ealumno@ucm.es
Enviando correo a alumno... Enviado.
Registrado usuario 'alumno'
```

Ilustración 92 - Ejemplo de ejecución del script registeruser

Crear Grupo

Si el campo "Grupo" recibido en el correo electrónico es "Creado", acudir directamente al paso siguiente, *Crear Grader y vincular a grupo*.

```
createGroup.php [options]

-n --nickname       Nickname del profesor
-g --group          Nickname del grupo
```

Es necesario haber realizado previamente el paso anterior, ya que de lo contrario, no se encontrará el *nick* del usuario, y la ejecución finalizará con un error.

El grupo se creará por defecto como "Privado".

Ejecución:

```
alvaro@alvaro-linux:/var/www/bolotweet/scripts$ php createGroup.php -nalvaro -ggmu
Registrado grupo 'gmu'.
Vinculado usuario 'alvaro' con grupo 'gmu'.
```

Ilustración 93 - Ejemplo de ejecución script createGroup

Crear Grader y vincular a grupo

Tanto si el grupo es nuevo, como si está creado, es necesario vincular el usuario al grupo en cuestión, tanto como profesor, como miembro del mismo.

```
createGrader.php [options]

  -d --delete         Borrar rango y desvincular usuario de grupo
  -i --id             ID del usuario
  -n --nickname       Nickname del usuario
  -g --group          Nickname del grupo
  -G --group-id       ID del grupo
```

Si el usuario a vincular ya tiene el rango de Grader, simplemente se le vinculará al grupo.

Ejecución:

```
alvaro@alvaro-linux:/var/www/bolotweet/scripts$ php createGrader.php -nalvaro -ggmu
Granting role 'grader' to user 'alvaro' (168)...OK
Granting role 'deleter' to user 'alvaro' (168)...OK
Vinculando usuario 'alvaro' con Grupo 'gmu' (17)...OK
```

Ilustración 94 - Ejemplo de script createGrader

Dar de alta alumnos de un grupo

Cuando se le informe al profesor sobre el éxito en la creación del usuario y del grupo, éste deberá enviar al Administrador un fichero **.xls*, o **.xlsx* con la lista de alumnos.

Para dar de alta a los usuarios, hay que hacer 2 sencillos pasos:

Convertir fichero XLS a CSV

Nombre del Script: convertirCSV.sh

Formato obligatorio del fichero (**.xls*, *xlsx*):

- El nombre del fichero debe ser el *nickname* del grupo.
- La primera línea debe ser el nombre de la asignatura.
- La segunda línea debe contener el nombre de las columnas.
- El fichero debe contener 4 columnas con el siguiente orden:
  apellido1 -- apellido2 -- nombre – email
- Cada usuario estará en una nueva línea.

Modo de ejecución: ./convertirCSV.sh nombreFicheroXLS

Ejecución:

```
alvaro@alvaro-linux:/var/www/bolotweet/scripts$ ./convertirCSV.sh gmu.xls
convert /var/www/bolotweet/scripts/gmu.xls -> /var/www/bolotweet/scripts/gmu.csv using Text - txt - csv (StarCalc)
```

Ilustración 95 - Ejemplo de ejecución script convertirCSV

Registrar usuarios y unirlos al grupo

Nombre del Script: registrarListas.sh

Formato obligatorio del fichero (**.csv*):

- Cada usuario estará en una nueva línea.
- Cada línea tendrá 4 campos en el siguiente orden:
  - *apellido1, apellido2, nombre, email*

Modo de ejecución: ./registrarListas.sh nombreFicheroCSV [nickGrupo]

Con este script es posible registrar una lista de usuarios y no insertarlos en ningún grupo.

Cada nuevo usuario que se registre recibirá un correo electrónico de bienvenida con indicaciones.

Si algún usuario ya estaba registrado, simplemente se le unirá al grupo, en el caso de haberlo indicado.

Ejecución:

```
alvaro@alvaro-linux:/var/www/bolotweet/scripts$ ./registrarListas.sh gmu.csv gmu
El usuario 'nombre1' ya existe.
'nombre1' unido al grupo 'gmu'
----------------------------
Registrado usuario 'nombre2'
Enviando correo a nombre2... Enviado.
'nombre2' unido al grupo 'gmu'
----------------------------
Registrado usuario 'nombre3'
Enviando correo a nombre3... Enviado.
'nombre3' unido al grupo 'gmu'
----------------------------
Registrado usuario 'nombre4'
Enviando correo a nombre4... Enviado.
'nombre4' unido al grupo 'gmu'
----------------------------
```

Ilustración 96 - Ejemplo de ejecución script registrarListas

Administrar grupos

Unir usuario a grupo

```
joingroup.php [options]

  -i --id             ID del usuario a unir al grupo
  -n --nickname       Nickname del usuario a unir al grupo
  -g --group          Nickname del grupo
  -G --group-id       ID del grupo
```

Ejecución:

```
alvaro@alvaro-linux:/var/www/bolotweet/scripts$ php joingroup.php -nalumno -ggmu
'alumno' unido al grupo 'gmu'
```

Ilustración 97 - Ejemplo de ejecución script joinGroup

Eliminar usuario de grupo

```
leavegroup.php [options]

  -i --id             ID del usuario a eliminar del grupo
  -n --nickname       Nickname del usuario a eliminar del grupo
  -g --group          Nickname del grupo
  -G --group-id       ID del grupo
```

Ejecución:

```
alvaro@alvaro-linux:/var/www/bolotweet/scripts$ php leavegroup.php -nalumno -ggmu
'alumno' ya no forma parte del grupo 'gmu'
```

Ilustración 98 - Ejemplo de ejecución script leaveGroup

Convertir en administrador de grupo

```
makegroupadmin.php [options]

  -g --group    Nickname del grupo
  -n --nickname Nickname del usuario a convertir en Admin
```

Ejecución:

```
alvaro@alvaro-linux:/var/www/bolotweet/scripts$ php makegroupadmin.php -nalumno -ggmu
'alumno' ahora es admin del grupo 'gmu'
```

Ilustración 99 - Ejemplo ejecución script makeGroupAdmin

Borrar Grupo

```
deletegroup.php [options]

  -i --id             ID del grupo
-n –nickname          Nickname del grupo
  -y --yes            No solicita confirmación
```

Ejecución:

```
alvaro@alvaro-linux:/var/www/bolotweet/scripts$ php deletegroup.php -ngmu
About to PERMANENTLY delete group 'gmu' (15). Are you sure? [y/N] y
Deleting...DONE.
```

Ilustración 100 - Ejemplo de ejecución script deletegroup

También puede borrar cualquier grupo directamente desde la interfaz:

Ilustración 101 - Ejemplo de borrado de grupo desde interfaz

Backup / Restore

Copia de Seguridad (BackUp)

Antes de cualquier actualización del sistema, es conveniente realizar una copia de seguridad. Para facilitar la tarea, se ha automatizado por completo el respaldo, de manera que ejecutando un script, es posible crear un fichero comprimido con una copia completa del sistema.

El script genera un fichero comprimido con todas las fuentes de BoloTweet, así como un respaldo de la BBDD de datos en el momento en que se realiza el *backup*.

*** En el proceso se solicitará la contraseña de root, y de la base de datos MysQL.*

Nombre del Script: backupBT.sh

La ejecución del script está completamente comentada, pudiendo ver en cada momento cómo va el proceso:

```
alvaro@alvaro-linux:/var/www/bolotweet/scripts$ ./backupBT.sh
Este script necesita ejecutar algunos comandos como SuperUsuario.
Presiona [Enter] para empezar el BackUp...
Introduce tu contraseña:
Parando Apache...Listo!
Creando Backup de /var/www/...Listo!
Creando Backup de la Base de Datos...
Enter password:
Listo!
Creando Fichero Final...Listo!
Borrando archivos temporales...Listo!
Iniciando Apache...
Listo!
BackUp terminado

Presiona [Enter] para terminar...
```

Ilustración 102 - Ejemplo de ejecución de script backupBT

Restauración del BackUp (Restore)

Si en algún momento se necesita restaurar una copia de seguridad, también se ha desarrollado un script para automatizar la tarea por completo, restaurando y levantando el sistema con la última versión.

Como es lógico, es necesario que el fichero con la copia de seguridad haya sido creado con el script *backupBT.sh*.

Nombre del Script: restoreBT.sh
Modo de ejecución: ../restoreBT nombreFichero

Borrar Usuario

```
deleteuser.php [options]

  -i --id             ID del usuario
  -n --nickname       Nickname del usuario
  -y --yes            No solicita confirmación
```

Ejecución:

```
alvaro@alvaro-linux:/var/www/bolotweet/scripts$ php deleteuser.php -nalumno
About to PERMANENTLY delete user 'alumno' (187). Are you sure? [y/N] y
Deleting...DONE.
```

Ilustración 103 - Ejemplo de ejecución script deleteUser

Si se dispone del rango "Administrador", o "Moderador", también es posible eliminar usuarios desde la interfaz gráfica.

Ilustración 104 - Ejemplo de borrado de usuario desde interfaz

*** La opción Silenciar mantendrá el usuario en el sistema, pero ninguno de sus tweets serán visibles.*

Borrar Mensajes

Un usuario con rango Moderador, o Administrador, puede borrar absolutamente cualquier *tweet*, sea de quien sea.

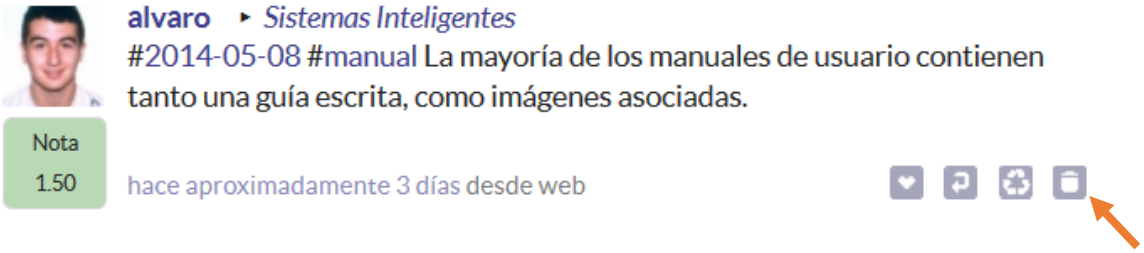


Ilustración 105 - Ejemplo de borrado de mensaje de manera gráfica

Asignar roles

A pesar de que las principales necesidades están automatizadas con los scripts comentados anteriormente, si por algún casual es preciso asignar roles manualmente, se puede hacer a través del siguiente script:

```
userrole.php [options]

  -d --delete         Elimina el rol
  -i --id             ID del usuario
  -n --nickname       Nickname del usuario
  -r --role           Rol a añadir (o borrar)
```

Algunos de los roles más relevantes son:

- grader → Es el rol que identifica a los profesores.
- administrator → Indica el administrador global del sistema.
- moderator → Ese usuario puede moderar el sistema.
- deleter → Indica que ese usuario puede borrar sus tweets.
- maxDeleter → Ese usuario puede borrar cualquier tweet.

Ejecución:

```
alvaro@alvaro-linux:/var/www/bolotweet/scripts$ php userrole.php -nalumno -rgrader
Granting role 'grader' to user 'alumno' (187)...OK
```

Ilustración 106 - Ejemplo de ejecución de script userRole

Desde la interfaz gráfica, también es posible asignar algunos roles relacionados con el sistema:

Ilustración 107 - Ejemplo de asignación de roles de manera gráfica

Enviar correos

Correo personalizado simple

```
echo "body" | sendemail.php [options]

-i --id               ID del usuario a enviar el correo
  -n --nickname       Nickname del usuario a enviar el correo
     --subject        Asunto del correo (Obligatorio)
```

El cuerpo del mensaje se escribirá en el comando con un *echo*, y posteriormente se ejecutará el script.

*** Este script únicamente sirve para mensajes individuales. Si se necesita otra opción, utilizar el script enviarEmail.php*

Ejecución:

```
alvaro@alvaro-linux:/var/www/bolotweet/scripts$ echo -e "Esto es un mensaje de prueba.\nEspero que
todo salga bien.\n\nUn saludo,\nAlvaro" | php sendemail.php -nalvaro --subject "Correo de prueba"
Sending to alvaro@prueba.es... done
```

Ilustración 108 - Ejemplo de ejecución de script sendEmail

Correo personalizado complejo

Si se desea enviar un correo personalizar con un contenido más complejo, con imágenes, con enlaces… es necesario utilizar este script. Además, permite enviar correos a un usuario, a un grupo, o a todos los miembros de BoloTweet.

```
enviarEmail.php [options]
  -i --id       ID of the user to send email
  -n --nickname nickname of the user to send email
  -g --group    Nickname or alias of group to send email
  -G --group-id ID of group to send email
  -a --all      Send email to all members
```

Para personalizar el correo, se debe editar internamente la función *body*, con el contenido que se quiera, y el asunto en la variable *subject*. Una vez realizado, tan sólo deberá seleccionarse una opción de envío.

Correo de bienvenida

Al utilizar el script *registrarListas.sh*, o *registeruser.php* para dar de alta a usuarios, automáticamente se les envía un correo de bienvenida con instrucciones.

No obstante, si se desea por alguna razón enviar ese mismo correo de bienvenida a algún usuario, o grupo en concreto, se puede hacer a través del siguiente script:

```
emailBienvenida.php [options]

  -i --id              del usuario al que mandar el correo
  -n --nickname       Nickname del usuario al que mandar el correo
  -g --group          Nickname del grupo al que mandar el correo
  -G --group-id       ID del grupo al que mandar el correo
  -a --all            Enviar correo a todos los miembros
```

Configurar BoloTweet

El administrador de BoloTweet podrá modificar las opciones de configuración del sistema de manera gráfica, a través del panel de administración.

Ilustración 109 - Panel de Administración

En la opción **Sitio**, es posible cambiar el nombre general de la página, zona horaria e idioma predeterminado del sistema, logo, y ajustar el límite de caracteres de los tweets.

En **Usuario**, es posible establecer una longitud máxima a la biografía de los usuarios, un mensaje de bienvenida personalizado, y habilitar o deshabilitar las invitaciones.

En **Acceder**, es posible configurar el acceso a la web (*Privado*, *Invitaciones*, *Cerrado*).

En **Rutas**, se puede configurar la URL del servidor, del sitio web, si se desean utilizar URL "*agradables*", se puede especificar un servidor especial para el tema, para las imágenes, para los archivos adjuntos, y se podrá modificar la configuración SSL.

En la opción **Sesiones** se puede configurar el manejo de las sesiones en BoloTweet.

En **Mensaje de Sitio** se puede editar el mensaje que se muestra a lo ancho de la página.

En **Licencia** se puede fijar el tipo de licencia que se quiera, así como información sobre la misma.

En **Plugins** se pueden habilitar o deshabilitar de manera gráfica algunos de los plugins existentes en *BoloTweet*.

D. Diario Beca de Colaboración

Esta sección está orientada a mostrar de manera más técnica la implementación y el desarrollo del sistema, pudiendo acudir en cualquier momento a una tarea concreta, para poder ver con más detalle cómo se ha desarrollado.

Para estructurar de manera correcta el documento, se ha divido en tres secciones principales, **Tareas**, **Bugs** y **Plugins**.

Las *Tareas* se corresponden con todos aquellos cambios o mejoras que se han introducido en el sistema, y que no están directamente relacionados con alguno de los plugins desarrollados, ni con bugs internos de *StatusNet*.

En el apartado *Bugs*, se comentarán los errores internos descubiertos en el *core* de *StatusNet*, y cómo se han corregido.

Por último, todos los nuevos plugins incluidos en BoloTweet 2.0 se explicarán con detalle en la sección *Plugins*.

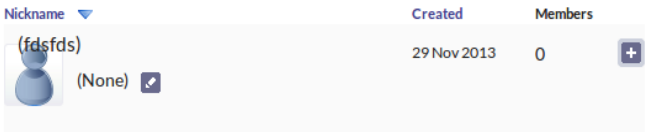
Bugs

Creación de grupo con nombre no válido

Ficheros modificados: newgroup.php

En el caso de crear un grupo con un nombre inválido, ya sea vacío, o con caracteres no permitidos, a pesar de mostrarse un error, se creaba el grupo con valor *null*.

Esto provocaba comportamientos inesperados en todas aquellas funciones que listaban, o realizaban consultas sobre los grupos existentes.



Solución:

Al generarse la excepción provocada por un nombre inválido, no se detenía la ejecución de la función, provocando que el grupo se creara de todas formas.

A continuación se muestra el código de la parte encargada de capturar las excepciones en el *parseo* del nombre del grupo:

```
try {
 $nickname = Nickname::normalize($this->trimmed('newnickname'));
        } catch (NicknameException $e) {
        $this->showForm($e->getMessage());
}
```

Si se compara el bloque *catch* con el resto, se puede ver como en éste, no se termina la ejecución al capturar la excepción.

Por tanto, la solución consiste en colocar un *return* dentro del bloque catch, de manera que una vez capturada la excepción, se muestre un mensaje de error y se termine la ejecución.

```
try {
$nickname = Nickname::normalize($this->trimmed('newnickname'));
        } catch (NicknameException $e) {
        $this->showForm($e->getMessage());
        return;
        }
```

Usuarios con email provocan que el avatar desaparezca

Ficheros modificados: <Ninguno>

Cuando se insertaba un usuario con correo electrónico asociado, o bien si se añadía un correo electrónico a un usuario ya registrado, su avatar se dejaba de mostrar.

Solución:

Tras depurar la ejecución para ver de dónde provenía el error, se descubrió que el causante era el *pluginGravatar*. Este plugin, posibilita la opción de integrar este servicio con *StatusNet*, permitiendo así a los usuarios utilizar su avatar personalizado de *Gravatar*. Sin embargo, el plugin no está bien desarrollado, de manera que cuando utiliza el *hookonEndProfileGetAvatar*, utiliza la siguiente condición:

```
if (empty($avatar)) {
       $user = $profile->getUser();
       if (!empty($user) && !empty($user->email)) { … }
```

Esto provoca que si el usuario no es vacío, y además tiene un correo electrónico asociado, se sustituye directamente la URL del avatar por:

```
$url="https://secure.gravatar.com/avatar.php?gravatar_id=".
       md5(strtolower($email))."&default=".
       urlencode(Avatar::defaultImage($size))."&size=".$size;
```

No realiza comprobaciones de si ese email está vinculado con alguna cuenta de *Gravatar*, o si tiene un avatar asociado, simplemente la reemplaza. Esto lógicamente provoca URL inexistentes, y por tanto avatares nulos.

Se podría mejorar el *plugin*, y realizar todo este tipo de comprobaciones, pero no hemos considerado necesario ni conveniente el uso de este *plugin* en *BoloTweet*, por lo que ha sido desactivado desde el panel de Administración, solucionando así este problema.

Si se quisiera deshabilitar desde línea de comandos, se debe utilizar el script *setconfig.php*, de la siguiente manera:

```
php setconfig.php 'plugins' 'disable-Gravatar' 1
```

Error al cancelar una petición de unión a un grupo

Ficheros modificados: Group_join_queue.php

Cuando un grupo está definido como *Privado*, es posible unirse al grupo, quedando pendiente una solicitud que el administrador del grupo deberá aprobar o rechazar.

Esta solicitud puede ser cancelada por el usuario que solicitó la unión al grupo en el momento que se quiera.

El error interno sucedía cuando un usuario intentaba cancelar la petición de unión al grupo, ya que el sistema no realizaba ninguna acción.

Solución:

Lo primero que se ha tenido que hacer, igual que en el resto de casos, es averiguar qué archivo está relacionado con este error, para posteriormente intentar corregirlo. En este caso, la clase *Group_join_queue*, encargada de tratar estas solicitudes, maneja la cancelación a través de la función *abort()*.

Esta función contenía la siguiente condición:

```
if ($request) { … }
```

Tras estudiar el código, se pudo observar como esa variable *$request* no existía en ningún lado, por lo que en ningún caso se ejecutaba el código interno de la condición, y por tanto nunca se llegaba a borrar la solicitud.

Para corregirlo, simplemente se ha tenido que eliminar esa comprobación, y ejecutar directamente el código de su interior, quedando de la siguiente manera:

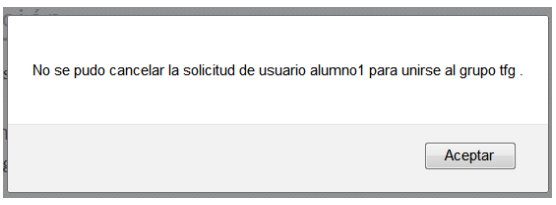
```
function abort()
    {
        $profile = $this->getMember();
        $group = $this->getGroup();

             if(Event::handle('StartCancelJoinGroup', array($profile, $group))) {
                $this->delete();
             Event::handle('EndCancelJoinGroup', array($profile, $group));
}
    }
```

En la clase *CancelsubscriptionAction*, antes de llamar a la función *abort()* se comprueba la validez del usuario, de la solicitud y del grupo, por lo que si se llega a ejecutar esta función, es porque todo es válido, y por tanto la comprobación de la variable $*request* no es necesaria.

Error al aceptar una solicitud de unión a un grupo

Ficheros modificados: Group_join_queue.php

Cuando un administrador de grupo aceptaba una solicitud de un usuario para unirse a un grupo, se mostraba un error interno, a pesar de que la solicitud sí era aceptada.



Solución:

De nuevo en el fichero *Group_join_queue.php*, encargado de tratar las solicitudes de unión a los grupos, existía otro error, en este caso en la función *complete().*

Cuando se utilizaba el *hookEndJoinGroup* y *StartJoinGroup*, se habían alterado de orden los parámetros, de manera que cuando luego otra clase (en este caso el *pluginActivity*), recibía los

parámetros, e intentaba acceder a funciones de cada uno, comenzaban a saltar excepciones.

Para solucionarlo, simplemente basta con invertir el orden de los parámetros, quedando de la siguiente manera:

```
function complete()
    {
        $join = null;
        $profile = $this->getMember();
        $group = $this->getGroup();
        if (Event::handle('StartJoinGroup', array($profile, $group))) {
            $join = Group_member::join($group->id, $profile->id);
            $this->delete();
            Event::handle('EndJoinGroup', array($profile, $group));
        }
        if (!$join) {
            throw new Exception('Internal error: group join failed.');
}
        $join->notify();
        return $join;
    }
```

Error al modificar aliases en un grupo

Ficheros modificados: editgroup.php

Cuando se modificaban los *alias* de un grupo, y se pulsaba el botón enviar, se mostraba un error interno que no permitía la modificación de los mismos.

Este error también sucedía cuando se enviaba un formulario de edición de grupo, sin haber modificado ningún dato.

Solución:

La función que maneja la modificación de un grupo es *trySave*, del fichero *editgroup.php.*

Esta función, cuando se envía un formulario de edición de grupo, obtiene todos los campos del formulario, y realiza un *update* a la base de datos. Si este *update* devuelve 0, se lanzaba una excepción con un error interno, cuando esto no debe ser así.

Por otro lado, la parte de los *alias* es tratada de forma diferente al resto de campos del formulario, de manera que primero se realiza el *update* con el resto de valores, y a continuación con los *alias*.

El problema es que como éstos últimos se actualizan después del resto de campos, si en los otros campos no se ha modificado nada, el *update* devolverá 0, se lanzara la excepción, y nunca se llegarán a actualizar los *alias*.

La única manera en que se podían modificar los *alias*, era modificando además algún otro campo, para que el *update* no devolviera 0, y así llegar a esa parte.

Como este funcionamiento no es para nada el esperado, se ha tenido que diseñar una serie de comprobaciones que eviten este comportamiento.

Lo primero que se ha hecho es comprobar si los campos introducidos por el usuario coinciden exactamente con los del grupo almacenado.

```
if (($this->group->nickname == $nickname) && ($this->group->fullname == $fullname) && ($this->group->homepage == $homepage) && ($this->group->description == $description) && ($this->group->location == $location) && ($this->group->mainpage == common_local_url('showgroup', array('nickname' => $nickname))) && ($this->group->join_policy == $join_policy) && ($this->group->force_scope == $force_scope)) {... }
```

Si los campos coinciden, quiere decir que no se ha modificado nada, y por tanto no es necesario llamar a la función *update*. Además, de esta manera, siempre se va a llegar a tratar los *alias*.

Se han añadido más comprobaciones, de manera que si los *alias* coinciden exactamente con los que tenía el grupo, y tampoco se ha modificado nada de la parte anterior, se muestre un mensaje indicando que no se ha producido ningún cambio.

```
$newaliases = array_unique($aliases);
$oldaliases = $this->group->getAliases();
$diffAlias1 = array_diff($oldaliases, $newaliases);
$diffAlias2 = array_diff($newaliases, $oldaliases);
if ($part1 && (empty($diffAlias1) && empty($diffAlias2))) { … }
if (!empty($diffAlias1) || !empty($diffAlias2)) { .. }
```

Si por el contrario los alias varían, se actualizan sin problema.

Además, se ha diseñado de manera que puedan suceder todas las combinaciones, sin que se produzca un comportamiento inesperado, es decir:

- Puede que se modifique la primera parte, y los alias no.
- Que se modifiquen la primera parte y los alias.
- Que no se modifique nada.
- Que se modifiquen los alias sólo.

Al acceder a un grupo desde el menú lateral, no se queda seleccionado

Ficheros modificados: groupsnav.php

Si se accedía a los grupos de los cuales se es miembro, a través de los *menu items* situados en la barra vertical izquierda, no se quedaban seleccionados.

Solución:

La creación de estos *menu items* se lleva a cabo en el fichero *groupsnav.php*. En la función getItems, encargada de obtener los grupos a los que pertenece el usuario, se va creando un *array* de elementos con un nombre de acción asociado para cada uno de ellos.

La idea es que posteriormente, cuando se llame a la función que muestre estos *menu items*, si la acción actual coincide con la asociada al elemento, se seleccione. Sin embargo, el nombre de acción que se asociaba a estos elementos era erróneo, y por ello nunca se mostraban como seleccionados.

Tras ver qué nombre de acción era el que correspondía, simplemente ha habido que modificar el nombre de la acción asociada. En este caso, *placeholder* ha sido sustituido por *showgroup* en la función getItems, quedando de la siguiente manera:

```
function getItems()
    {
        $items = array();

        while ($this->groups->fetch()) {
            $items[] = array('placeholder',
                             array('nickname' => $this->groups->nickname,
                                   'mainpage' => $this->groups->homeUrl()),
                             $this->groups->nickname,
                             $this->groups->getBestName()
                            );
        }

        return $items;
}
```

Corregido script updateurls.php

Ficheros modificados: updateurls.php

Este script se encarga de actualizar las URL internas de la base de datos de los usuarios. Sin embargo, la implementación de este script no era correcta, y solamente se actualizaba la URL del perfil del primer usuario.

Solución:

Una vez comprendido el funcionamiento del script, se ha pasado a modificar por completo la obtención de todos los usuarios.
En esta ocasión, se ha realizado una consulta SQL, encargada de obtener todos los usuarios de la base de datos.

```
SELECT * FROM profile order by id asc
```

Una vez obtenidos los perfiles, se almacenan en un *array*. Posteriormente, utilizando un bucle *while*, este *array* se va recorriendo actualizando para cada usuario sus URL internas.

El código completo que se ha añadido para corregir el bug es el siguiente:

```
$qry = "SELECT * FROM profile order by id asc";
    $pflQry = new Profile();

    $pflQry->query($qry);

    $members = array();

    while ($pflQry->fetch()) {
        $members[] = clone($pflQry);
    }

    $pflQry->free();

    foreach ($members as $member) {

        $user = $member->getUser();

        printfv("Updating user {$user->nickname}...");
        try {
            $profile = $user->getProfile();

            updateProfileUrl($profile);
            updateAvatarUrls($profile);

            // Broadcast for remote users

            common_broadcast_profile($profile);
        } catch (Exception $e) {
            printv("Error updating URLs: " . $e->getMessage());
}
        printfv("DONE.");
    }
```

Tareas

Preparación entorno MySQL

Ficheros modificados: my.cnf

Para poder trabajar con *BoloTweet*, es necesario e imprescindible el trabajo directo y constante con MySQL.

Para poder visualizar los resultados de las consultas de manera correcta, es necesario modificar el parámetro *pager* de MySQL. Para ello, se ha tenido que editar el fichero:

/etc/mysql/my.cnf

Y debajo del apartado [mysql], se ha añadido la siguiente línea:

```
pager = less -inS
```

Enviar correos desde BoloTweet

Ficheros modificados: config.php

Para poder enviar correos electrónicos desde BoloTweet, ha sido necesario configurar un servidor de correo electrónico, en este caso, utilizando una cuenta de Gmail.

Se ha utilizado como referencia el archivo *config.php.sample*, donde se muestra un fichero de configuración de StatusNet de ejemplo.

El código necesario a incluir en el fichero *config.php* para configurar el envío de correos electrónicos en el servidor es el siguiente:

```
$config['mail']['backend'] = 'smtp';
$config['mail']['params'] = array(
'host' => 'smtp.gmail.com',
'port' => 587,
'auth' => true,
'username' => 'bolotweetv2@gmail.com',
'password' =>enElDocumento
);
```

Es por tanto con esta cuenta, y con esa dirección, con la que se envían todos los correos del sistema, tales como correos de bienvenida, recuperación de contraseña, actividad del sistema…

Deshabilitar la opción de eliminar cuenta para alumnos

Ficheros modificados: <Ninguno>

En el perfil de los usuarios existía la opción de eliminar la cuenta. Actualmente, y tal y como está planteado *BoloTweet*, esta opción no debe estar disponible, ya que el alta y la baja de usuarios es un tema plenamente administrativo.

Para solucionar esto, en el archivo *profilesettings.php*, encargado de mostrar todo lo relativo al perfil de los usuarios, se puede ver la siguiente comprobación en la función *showAside*:

```
if ($user->hasRight(Right::DELETEACCOUNT))
```

Esta comprobación lo que hace es que si el usuario tiene el derecho de borrar su cuenta, se muestre un enlace para ello en las opciones del perfil.

Si se va trazando el comportamiento de esta comprobación, en la función hasRight de *Profile*, que es a la que se llama desde *$user->hasRight*, se puede ver lo siguiente:

```
        case Right::DELETEACCOUNT:
$result = common_config('profile', 'delete');
       break;
```

En este caso lo que se hace es buscar a través de la función *common_config*, qué valor asociado tiene la opción [profile][delete]. Este valor puede estar definido, o bien en el fichero de configuración del servidor *config.php*, o bien en la tabla *config* de la base de datos.

Gracias a esta línea se puede deducir que el borrado de cuentas se puede tratar desde el archivo de configuración, por lo que si se mira en el fichero *CONFIGURE.txt*, donde se explican todas las posibles opciones de configuración del servidor, se puede leer lo siguiente:

> *delete: whether users can delete their own accounts. Defaults to false.*

A pesar de que por defecto debería estar la opción a *false*, la clase *PrivateSite* modifica la configuración del servidor dependiendo de la visibilidad del sistema. En este caso, como BoloTweet está configurado como privado, realiza lo siguiente:

```
'profile'    => array('delete' => 'true')
```

Se podría modificar ese valor, pero no es necesario *hardcodear* esta decisión, cuando puede y debe hacerse desde el fichero de configuración.

Para ello, lo que he hecho es utilizar el script *setconfig.php*, encargado de modificar la configuración del fichero config.php pero de manera interna, es decir, directamente en la base de datos.

```
php setconfig.php 'profile' 'delete' 0
```

Cambiar idioma a Español

Ficheros modificados: <Ninguno>

Uno de los requisitos necesarios de *Bolotweet*, era el cambio del idioma de la interfaz al español.

Tanto desde el perfil de administración, como en el perfil personal de cada usuario, es posible seleccionar el idioma que se desee. Sin embargo, esta opción no provocaba ningún cambio en el sistema.

Tras leer problemas similares en los foros de *StatusNet*, en uno de ellos se hablaba sobre la generación de archivos de tipo *.mo (Machine Object).

Para conocer más al respecto, se acudió al fichero *Readme.txt*, situado en la carpeta *locale* (directorio con todas las traducciones).

> *== Building runtime translations ==*
>
> *If you are working with a direct git checkout or have customized any message files, you will need to build binary .mo files from the .po source files for translations to work in the web app.*
> *If gettext and GNU make are installed, you can simply run 'make' in the main StatusNet directory, and all core and plugin localizations will be recompiled.*

En las subcarpetas de *locale* dedicadas a cada idioma, únicamente estaban los ficheros *.po.

Tras buscar información acerca de esto, resulta que los ficheros *.po (Portable Object) son ficheros generados por el traductor de *gettext* a partir de una plantilla *.pot. En este caso, esta plantilla se encuentra en la raíz de la carpeta *locale* (*statusnet.pot*).

Posteriormente, es necesario convertir estos objetos portables a ficheros binarios, utilizando la herramienta `msgfmt`.

Sin embargo, en el fichero *Makefile* de StatusNet, ya están predefinidas las reglas necesarias para generar esos ficheros *.mo, por lo que únicamente es necesario instalar los paquetes *make* y *gettext*, situarse en la raíz del proyecto, y a través de la terminal ejecutar el comando *make*.

Fichero *Makefile*:

```
# Warning: do not transform tabs to spaces in this file.

all : translations

core_mo = $(patsubst %.po,%.mo,$(wildcard locale/*/LC_MESSAGES/statusnet.po))
plugin_mo = $(patsubst %.po,%.mo,$(wildcard plugins/*/locale/*/LC_MESSAGES/*.po))

translations : $(core_mo) $(plugin_mo)

clean :
	rm -f $(core_mo) $(plugin_mo)

updatepo :
	php scripts/update_po_templates.php --all

%.mo : %.po
	msgfmt -o $@ $<
```

Una vez hecho esto, se comenzarán a generar los ficheros binarios, y ahora sí que se aplicarán las traducciones al seleccionar el cambio de idioma desde la interfaz.

Limpieza SPAM en servidor

Ficheros modificados: <Ninguno>

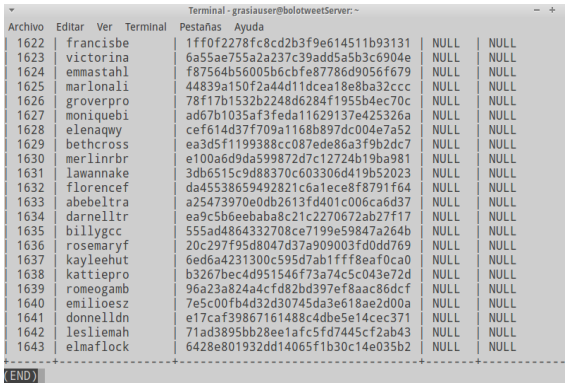
Una vez puesto en marcha el sistema en el servidor de GRASIA, se comenzaron a sufrir grandes ataques generados por *robots* que se registraban, y comenzaban a publicar mensajes con de todo tipo de *Spam*.

Para solucionar esto, lo primero que se hizo fue desde el perfil de administrador, cerrar y deshabilitar los registros.

Posteriormente, lo siguiente que se tuvo que realizar es la limpieza de todos esos usuarios, así como de todos sus mensajes.

La compleja estructura de la BBDD de StatusNet, provoca que a pesar de eliminar los *tweets* con *spam*, se sigan almacenando restos por otras tablas, por lo que se ha tenido que revisar manualmente toda la base de datos, para poder limpiar por completo BoloTweet.

Para borrar los usuarios, se ha utilizado el script *deleteuser.php*, encargado de borrar un usuario del sistema, así como sus tweets asociados. Para utilizar este script es necesario indicarle un ID de usuario, por lo que a través de MySQL, se localizaron todos los ID en la tabla *Users*.



A continuación se desarrolló un *mini* script bash para automatizar el borrado de todos estos usuarios, quedando de la siguiente manera:

```
#!/bin/bash
for ((c=6; c<=1643; c++))
do
php deleteuser.php -i$c -y
done
```

Sin embargo, en el resto de tablas quedaron restos de estos usuarios y de sus acciones. Como no existe ningún script que se encargue de limpiar estos rastros, ha sido necesario realizar la limpieza desde MySQL manualmente.

Se han borrado restos de las tablas:

- conversation
- deleted_notice
- file (había spam de youtube)
- file_oembed (video de Youtube SPAM)
- file_redirection
- file_thumbnail
- inbox

Quitar la opción Invite Colleagues

Ficheros modificados: <Ninguno>

Por defecto, StatusNet dispone de la posibilidad de ofrecer invitaciones al sistema para nuevos usuarios.

Sin embargo, este tema es administrativo, siendo el administrador del sistema quien dará de alta o de baja a los usuarios a petición de los profesores.

Por esta razón, no tiene sentido que se puedan ofrecer invitaciones, y que exista un botón destinado a ello. Para eliminar esta opción, StatusNet ofrece la posibilidad de tratarlo a través del fichero de configuración del sistema.

Como en otras ocasiones, se debe acudir al fichero *CONFIGURE.txt* para conocer cómo modificar esta opción:

invite

For configuring invites.
enabled: Whether to allow users to send invites. Default true.

Para variar este valor, se ha utilizado de nuevo el script *setconfig.php*

```
php setconfig.php 'invite' 'enabled' 0
```

Con esto se deshabilita la posibilidad de invitar, y automáticamente se quita el botón destinado para ello.

Letras solapadas en la versión móvil

Ficheros modificados: mp-screen.css

BoloTweet está preparado para funcionar en dispositivos móviles con una interfaz ligeramente diferente. Sin embargo, con el aumento de nuevas opciones y características no todo se muestra como debería.

En la barra de arriba, al añadir la opción de *Grade Reports*, todas las opciones no caben en una sola línea, provocando que se solape parte del contenido inferior.

Para solucionar esto, se debe modificar el plugin *MobileProfile*, que es el encargado de tratar la visualización del sistema en dispositivos móviles.

Por ello, ha sido necesario editar sus archivos CSS, añadiendo y modificando algunas reglas, para que la visualización sea la esperada.

```
#aside_primary_wrapper {
   margin-top: 20px;
}

address img + .fn {
margin-top: 20px;
}


#header-search #search-q {

margin-bottom: 3px;

}
```

El resultado final obtenido es el siguiente:

Configurar BoloTweet con conexión obligatoria por HTTPs

Ficheros modificados: statusnet.php, util.php

El paquete StatusNet ofrece opciones para controlar el uso de HTTPs en el sistema. Sin embargo, en el servidor no funcionaban correctamente, provocando redirecciones infinitas, problemas con scripts, con ficheros CSS…

Ya que por tanto, la opción para trabajar con SSL no funciona como se espera, ha sido necesario modificar el *core* internamente.

En primer lugar, se ha tenido que modificar la función isHTTPS del fichero *statusnet.php*, para obligarla a devolver en todo caso *true*, que es lo que se requiere.

```
static function isHTTPS() {
return true;
}
```

El siguiente paso es obligar a que la URL sea siempre HTTPS, por lo que ha sido necesario modificar la función *common_path*, del fichero *util.php*. En este caso, lo que se ha hecho es obligar a que el protocolo sea siempre *https*, evitando así comportamientos inesperados.

```
$proto = 'https';
return $proto.'://'.$serverpart.'/'.$pathpart.$relative;
```

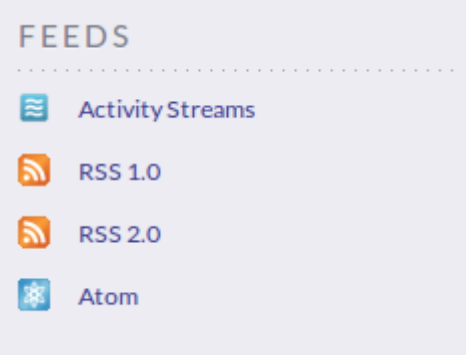
Como es esta función la que utilizan absolutamente todas las clases y los métodos del sistema para generar las direcciones, nos aseguramos que en todo momento se generen urls con HTTPs.

Un detalle importante es que ha sido necesario de nuevo limpiar el sistema por completo, tal y como se ha explicado en la tarea *Limpieza SPAM en servidor*.

Esto es porque cuando se crea un usuario, un tweet, un grupo… se le asocian URIs internas, que se almacenan en la base de datos. Por ello, al modificar por completo el protocolo, todas las URIs internas se habían almacenado con http, provocando accesos fallidos a estos recursos.

Eliminar FEEDS de BoloTweet

Ficheros modificados: action.php



Para eliminar los *feeds* del sistema, se ha tenido que modificar la función showExportData(), del fichero *action.php*, eliminando todo su contenido, y evitando así que se muestre el apartado con los FEEDS.

```
/**
     * Show export data feeds.
     *
     * @return void
*/
    function showExportData()
    {
    }
```

Problema con algunas URL en GRASIA, apuntan a 127.0.0.1

Ficheros modificados: <Ninguno>

Al realizar todas las pruebas en local, cuando se traspasó por primera vez la base de datos a GRASIA, las URIs internas almacenadas no eran válidas, ya que contenían como dirección base del servidor 127.0.0.1.

Por ello, ha sido necesario borrar de la base de datos de GRASIA todas aquellas tablas con url incorrectas, o que sean dependencias de tablas con url incorrectas.

Para llevar esto a cabo, se ha tenido que revisar manualmente qué tablas son las afectadas.

- conversation
- deleted_notice
- file
- file_redirection
- grades
- inbox
- profile_role
- user_location_prefs

Para borrar las tablas por completo desde MySQL, se ha utilizado el comando:

```
delete from nombreTabla;
```

Ahora el siguiente paso ha sido crear de nuevo los usuarios y grupos de prueba. Sin embargo, como algunas tablas están definidas con un campo *autoincrement*, los nuevos usuarios no van a comenzar desde 0, tal y como se espera.

Para corregir esto, se debe hacer un *truncate* a las tablas que dispongan de tal campo, reiniciándolas por completo.

Se ha aplicado por tanto este comando a las siguientes tablas:

- grades;
- conversation;
- file;
- message;

- notice;
- oauth_application;
- profile;
- profile_detail;
- profile_list;
- queue_item;
- user_group;

Una vez hecho esto, es necesario crear todos los usuarios del sistema, empezando por el Administrador global, que ha sido eliminado, y posteriormente los usuarios y profesores de prueba.

Para hacer esto, se han hecho uso de los scripts proporcionados por StatusNet.

Para registrar los usuarios:

```
php registeruser.php -nNombre –wPassword
```

Para asignar los rangos:

```
php userrole.php -iid -rrole
```

Al administrador del sistema es necesario aplicarle los rangos *owner*, *moderator* y *administrator*. Para los profesores, el rango a aplicar es *grader*.

Con esto se ha conseguido reiniciar las tablas de la base de datos del servidor, e iniciarla con usuarios de prueba para comenzar su uso.

```
+----+-----------+----------------------------------+-------+-----------
| id | nickname  | password                         | email | incomingem
+----+-----------+----------------------------------+-------+-----------
|  1 | adminbt   | cafd7d77b7dbd68876d80ab8baba3e3b | NULL  | NULL
|  2 | alumno1   | 7cf22a9c4e21e584a07e7d162b94257a | NULL  | NULL
|  3 | alumno2   | dcaf135adb452ab48bac68199852b08d | NULL  | NULL
|  4 | profesor1 | 6062833186fdf539b23a2f0192e990f3 | NULL  | NULL
|  5 | profesor2 | 8a23563a9d3cde023bb8bb5d1b3cce32 | NULL  | NULL
|  6 | alvaro    | dcc4c191d908b5e4f94794000d689dd5 | NULL  | NULL
+----+-----------+----------------------------------+-------+-----------
(END)
```

Modificar y crear traducciones al Español

Ficheros modificados: statusnet.po, statusnet.pot

Algunas de las traducciones por defecto al español que incluye StatusNet no son correctas, o no tienen el sentido que se quiere dar con BoloTweet.

Para corregir esto, hay que localizar el fichero */locale/es/LC_MESSAGES/statusnet.po*, y posteriormente las traducciones que se desean cambiar.

La estructura de las traducciones es la siguiente:

#. TRANS: Comentario acerca de la localización de esta cadena
msgid "Texto original"
msgstr "Texto traducido"

Si se desea modificar una traducción existente, debe editarse el campo *msgstr*.

En algunos casos ha sido necesario añadir traducciones inexistentes en el fichero de traducción, requiriéndose unos pasos más complejos.

Lo primero que hay que hacer es editar la plantilla de traducción (*statusnet.pot*), para añadir información con una estructura determinada sobre la nueva cadena a traducir.

#. TRANS: Comentario acerca de la localización de esta cadena
#: Archivo y línea donde se encuentra la cadena que queremos traducir
msgid "Texto original"
msgstr ""

Una vez modificada la plantilla, se debe editar el fichero *statusnet.po*, y añadir las líneas correspondientes a esta traducción con la estructura de la primera opción, y colocando en *msgstr* la traducción que corresponda.

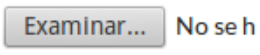
Una vez modificadas todas las traducciones que se precisen, para que se actualice el sistema, es necesario regenerar los ficheros binarios. Para ello, hay que eliminar el fichero *statusnet.mo*, y ejecutar el comando *make* en la raíz del proyecto.

De esta manera, el fichero binario contendrá las últimas traducciones, y ya serán visibles en la interfaz de BoloTweet.

Formulario cortado en carga de imágenes

Ficheros modificados: base/css/display.css

Cuando se accedía al formulario para cargar imágenes de avatar, por defecto se mostraba cortado, siendo imposible leer su contenido.



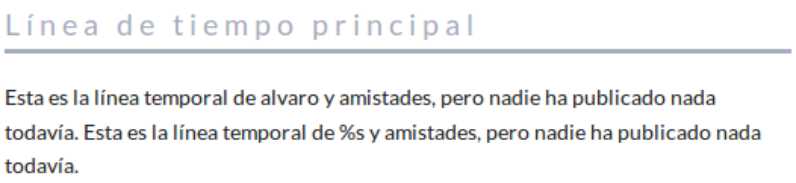
Para solucionarlo, se ha tenido que editar el fichero CSS que controla este elemento, añadiendo una nueva regla:

```
.form_settings .form_data li#settings_attach input {
width:auto;
}
```

Error en el mensaje de la línea personal vacía

Ficheros modificados: statusnet.po, all.php

Al acceder una línea personal vacía, se mostraba un mensaje repetido, y con un mal formato.



Este error era producido por una mala traducción, por lo que el error estaba en el fichero *statusnet.po*.

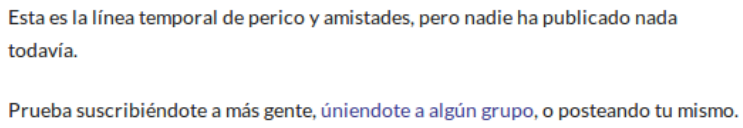
"Try subscribing to more people, [join a group](%%action.groups%%) or post "
"something yourself."
msgstr ""
"Esta es la línea temporal de %s y amistades, pero nadie ha publicado nada "
"todavía."

Como se puede ver, la cadena estaba traducida incorrectamente, por lo que se mostraba dos veces lo mismo.

La solución ha sido traducirlo manualmente al español, y regenerar los archivos de traducción.

Por otro lado, en el fichero actions/all.php, que es donde se muestra este mensaje, se han incluido algunos saltos de línea para que el resultado sea mejor.

"Try subscribing to more people, [join a group](%%action.groups%%) or post "
"something yourself."
msgstr ""
"Prueba suscribiéndote a más gente, [úniendote a algún grupo](%%action.groups%%), "
" o posteando tú mismo.



En el panel de Administración el título aparece cortado

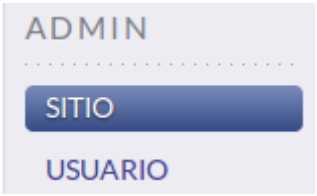
Ficheros modificados: statusnet.po

Como toda la interfaz, así como las reglas de visualización han sido desarrolladas basándose en el idioma inglés, algunas traducciones no quedan como deberían. Para solucionar esto, en ocasiones hay que modificar reglas CSS, y en otras, como este caso, y debido a que la estructura de la página no acepta un nombre tan largo, ha sido necesario modificar la traducción de ese campo.

Se ha modificado por tanto el fichero *statusnet.po*, tal y como se explicó en otra tarea anteriormente, y se ha cambiado el campo msgstr por Admin.

```
msgid  "Admin"
msgstr "Admin"
```

De esta manera, el significado es el mismo, y no se descuadra la interfaz de BoloTweet.



Al crear una encuesta, el autor debe votar para ver los resultados

Ficheros modificados: PollPlugin.php

El plugin que se encarga de las encuestas, *PollPlugin*, está desarrollado de manera que los resultados de una encuesta solamente pueden ser visualizados si se ha contestado a la misma.
Sin saber claramente si es un fallo de implementación, o una función deseada por algún motivo, se ha considerado que este comportamiento se debía corregir para BoloTweet.

No tiene sentido que si el profesor lanza una encuesta a los alumnos, deba contestarla él para poder comprobar los resultados.

Para poder encontrar la solución, es necesario estudiar el plugin, entender su funcionamiento, y encontrar la parte en que se maneja este aspecto.

Ha sido necesario modificar el fichero *plugins/poll/PollPlugin.php*, que es dónde se decide qué mostrar dependiendo de si se ha contestado o no la encuesta.

Esta comprobación estaba implementada de la siguiente manera:

```
if ($response)
```

Es decir, o bien se había contestado, o no se mostraban nunca los resultados. Para corregirlo, lo que se ha hecho es modificar la comprobación, y aprovechando que el objeto *poll*, tiene la ID del usuario que la ha creado, la comprobación final ha quedado así:

```
if ($response || ($user->id === $poll->profile_id)) ...
```

De esta manera, si se ha respondido la encuesta, o si el usuario actual es el creador de la misma, pueden observarse directamente los resultados.

Cualquier alumno puede crear una asignatura

Ficheros modificados: actionsgroupdirectory.php

Tal y como está planteado StatusNet, cualquier usuario tiene privilegios suficientes para crear un grupo. Esto no es ningún *bug*, pero sí es un comportamiento erróneo en BoloTweet.

Al tratar a los grupos como asignaturas, no tiene sentido que cualquier alumno pueda crear, editar, y modificar asignaturas a su antojo.

Para cambiar este comportamiento, se ha tenido que modificar el *core* interno del sistema. Un usuario solamente puede crear un grupo a través de un enlace mostrado en la opción *Groups*. Por tanto, si se restringe ese enlace únicamente a los profesores, estará solucionado.

El archivo *actionsgroupdirectory.php*, del plugin *Directory* es el que se encarga de crear este enlace. Para solucionarlo, se ha añadido en ese mismo fichero una comprobación con la condición de que el usuario actual sea *grader*, es decir, profesor.

```
if (common_logged_in() && common_current_user()->hasrole('grader'))
```

De esta manera, este enlace no estará visible a los alumnos, y se logrará el comportamiento deseado.

Error al enviar gran cantidad de eventos

Ficheros modificados: newevent.php, Notice.php

A pesar de BoloTwet utiliza plugins que dan funcionalidades específicas como Preguntas, Eventos, Encuestas… todas ellas son tratadas internamente como noticias, y por tanto se les aplican sus mismas restricciones.

De esta manera, la opción Evento, utilizada para representar y anunciar acontecimientos por parte de profesores, está sometida al igual que cualquier anotación al límite de 140 caracteres.

Sin embargo, un evento está compuesto por un título, un lugar, una fecha, una descripción… por lo que es muy común sobrepasar la limitación de tamaño, y no poder crear el evento deseado.

Un error muy similar a éste ha sido solucionado en la tarea *Exceso de tamaño al registrar listas de usuarios*, por lo que se solucionará utilizando la idea allí empleada.

En aquel caso, cuando un objeto del tipo *Activity* llega a la función Notice::saveNew, lleva un atributo *source* único, '*activity*'. Con este atributo, se ha podido realizar una comprobación de manera que si el objeto que recibe esta función tiene el atributo *source* de tipo 'activity', no se le aplicará la comprobación de límite de caracteres.

Utilizando esta idea, se ha recreado este comportamiento con el objeto Event. Sin embargo, en este caso por defecto no incluía ningún atributo *source*, por lo que se ha tenido que añadir.

Para ello, se ha modificado el fichero */plugins/events/newevent.php* de manera que al llamar a la función que guarda el evento, el objeto ya incluya el atributo *source* identificativo.

```
$options['source'] = 'event';
```

Posteriormente, lo único que hay que hacer es añadir una nueva excepción en la función saveNew de Notice, de manera que si el objeto es de tipo 'event', tampoco se le aplique la comprobación de límite de caracteres:

```
if($source!='activity' && $source!='event'){ … }
```

De esta manera se ha conseguido el comportamiento esperado, permitiendo además un esquema de exclusiones realmente sencillo, en el caso de que otros nuevos objetos o funciones no requieran comprobación de tamaño.

Cuando se accede a un grupo privado no se muestra ninguna información

Ficheros modificados: showgroup.php

El sistema de grupos de *BoloTweet* permite visualizarlos como públicos o privados. Cuando un grupo está configurado como privado, además de necesitar aprobación la unión al grupo, sus mensajes solamente son visibles para sus miembros.

Provocado por este comportamiento, cuando un usuario accede a un grupo privado del cual no es miembro, únicamente aparece el grupo vacío, sin indicación de ningún tipo. Este comportamiento puede llevar a confusiones, ya que en ningún momento se indica que el grupo es privado.

Para solucionar esto, se ha añadido una nueva comprobación en el fichero *actions/showgroup.php*, de manera que si el grupo es privado, y el usuario que ha entrado en él no pertenece al mismo, se le muestre un mensaje con información.

```
if ($this->group->force_scope &&
   (empty($this->userProfile) || !$this->userProfile->isMember($this->group))) {
      $this->elementStart('p');
      $this->raw('Este grupo es privado, por lo que no es posible ver su
      contenido.<br/><br/>' .
      'Si lo desea, puede comunicarse con el administrador del grupo, o solicitar
      unirse al grupo. ');
      $this->elementEnd('p');

}
```

Cuando se accede a un grupo sin mensajes no se muestra ninguna información

Ficheros modificados: showgroup.php

En los grupos sin mensajes, tanto si eres miembro como si no, en ningún momento se muestra ninguna información acerca de qué ocurre, tan sólo se ve el contenido del grupo vacío.

Esto puede provocar confusiones, pues en un principio no se sabe si no hay mensajes, si el grupo es privado, si por alguna razón no se tiene privilegios para ver el contenido del grupo…

Para solucionar esto, en el archivo *actions/showgroup.php*, encargado de mostrar el contenido de los grupos, se ha añadido una nueva comprobación de manera que si el *array* de mensajes del grupo es vacío, se muestre un mensaje indicándolo.

```
if (empty($this->notice->_items) ) {
                $this->elementStart('p');
                $this->raw('¡Una lástima! Este grupo aún no tiene mensajes. ');
                $this->elementEnd('p');
            }
```

Corregidos errores menores

Ficheros modificados: statusnet.po, eventlistitem.php, NoticeTaskForm.php, poll.css, Event.po, Event.pot, pollresponseform.php, pollresultform.php

- Quitadas líneas de separación entre opciones de encuestas.
- El campo 'To:' de las tareas, cambiado por 'Para:'.
- En las encuestas se ha corregido el título, se mostraba dos veces.
- Corregidas diversas traducciones al español.

Desarrollar un script para poder registrar listas de usuarios

Ficheros modificados: convertirCSV.sh, registrarListas.sh

Uno de los aspectos más importantes en el nuevo BoloTweet es la administración del sistema. Hasta ahora, no existía ningún tipo de script ni herramienta que permitiera realizar una gestión completa del sistema desde el punto de vista del Administrador.

Por ello, era una necesidad la existencia de un script que se encargara automáticamente de registrar listas de usuarios, y posteriormente vincularlos a una asignatura concreta.

Para comenzar esta tarea, se tuvo que decidir una estructura común de las listas, y un proceso a seguir estructurado.

El siguiente paso ha sido decidir en qué lenguaje desarrollar el script, bien en *PHP*, o en *Bash*. Tras valorar pros y contras, se optó por implementarlo en *bash*, ya que su uso es más sencillo, y desde él se puede llamar muy fácilmente a otros scripts en PHP, por ejemplo para registrar los usuarios, o para mandarles un correo de bienvenida.

En este caso, el script tenía que ser capaz de recibir un fichero en Excel con una estructura determinada, convertirlo a un fichero en formato CSV, y posteriormente recorrerlo registrando a los usuarios y vinculándolos al grupo que correspondiera.

Para abordar la conversión al formato CSV, en un primer momento se utilizó la librería *xls2csv*, pero su soporte para ficheros únicamente *.xls, y no *.xlsx, así como su escasa posibilidad de configuración obligaron a optar por otra herramienta más completa, en este caso proporcionada por LibreOffice.

Esta herramienta permite convertir desde línea de comandos un fichero en formato *xls* o *xlsx* a *csv*, además de proporcionar una configuración muy personalizable.

```
libreoffice --headless --convert-to csv $1
```

Con este comando se obtiene un fichero CSV con una estructura correcta, de manera que cada línea representa a un alumno, con los campos separados por comas.

Un detalle muy importante es que el fichero Excel recibido no puede contener espacios, ni tabulaciones extra en los campos, ya que de lo contrario el fichero CSV generado no será correcto.

Una vez realizada la conversión, es necesario aplicarle dos cambios más al fichero generado.

El primero consiste en eliminar las dos primeras líneas del fichero CSV generado, que corresponden al nombre de la asignatura, y al nombre de las columnas, dejando así un fichero únicamente con alumnos.

Para ello, se ha utilizado el comando *sed*, con los siguientes parámetros:

```
sed -i "1,2d" $fullname
```

Como para utilizar el comando *sed* es necesario indicar el nombre del fichero, se ha utilizado el comando *cut* para separar el nombre de la extensión del archivo de entrada en formato Excel, y se le ha añadido la extensión ".csv", dando como resultado el nombre del fichero que se genera al realizar la conversión utilizando LibreOffice.

```
ext=".csv"
name=( $(echo $1 | cut -d '.' -f1) )
fullname=$name$ext
```

El segundo cambio viene ocasionado por fallos encontrados en las pruebas realizadas sobre ficheros CSV. Resulta que los nombres con acentos o caracteres especiales producían un comportamiento inesperado, de manera que si el nombre de un alumno era "JOSÉ", en BoloTweet únicamente se cargaba "JOS".

La primera solución que se planteó fue utilizar el comando *tr*, de manera que se reemplazaran las letras con acento, pero en el caso de las "ñ", por ejemplo, el resultado no era el deseado.

```
tr "ÁÉÍÓÚÑáéíóúñºªü" "AEIOUNaeiounoau"
```

Tras buscar alternativas, finalmente se optó por tratar la codificación del fichero.

Los archivos CSV generados tenían una codificación ISO-8859-1, mientras que en este caso, la ideal sería UTF-8, ya que soporta acentos y caracteres especiales sin producir ninguna salida extraña.

Por ello, el siguiente paso fue buscar cómo realizar esta conversión, utilizando finalmente el comando *iconv*:

```
iconv -f ISO-8859-1 -t UTF-8 -o $fullname $fullname
```

De esta manera, con el script *convertirCSV.sh*, se consigue generar a partir de un fichero Excel, un fichero CSV con únicamente una lista de alumnos, y en formato UTF-8.

Archivo Excel inicial (XLS, XLSX)

1	Trabajo de Fin de Grado			
2	**Primer apellido**	**Segundo apellido**	**Nombre**	**E-mail Universidad**
3	ORTEGO	MARCOS	ÁLVARO	alvorteg@ucm.es
4	APELLIDO1	APELLIDO2	NOMBRE1	nombre@ucm.es

CSV Generado

```
ORTEGO,MARCOS,ÁLVARO,alvorteg@ucm.es
APELLIDO1,APELLIDO2,NOMBRE1,nombre@ucm.es
```

En el Script se encuentra toda la información sobre cómo ejecutarlo correctamente, y qué parámetros necesita.

```
#!/bin/bash

# Este script está hecho para convertir los ficheros xlsx o xls con los usuarios
# a formato csv, para poder posteriormente registrarlos.
#
# Se debe respetar la siguiente estructura:
#
#   - El archivo tiene que tener la extensión xls o xlsx.
#   - La primera línea debe ser el nombre de la asignatura.
#   - La segunda línea debe ser el nombre de la columna.
#
# El script se ejecuta de la siguiente forma:
#       ./convertirCSV.sh nombreCSV
#
# Siendo nombreCSV el nombre del fichero con la lista de usuarios.
```

El siguiente paso es desarrollar otro script que utilice el fichero CSV generado, para posteriormente registrar los usuarios.

Este script se ha desarrollado de nuevo en *bash*, ya que se necesita realizar llamadas a otros scripts en *PHP*.

La idea principal de este script es recorrer el fichero CSV, de manera que cada campo separado por comas se almacene en una variable, que posteriormente servirán para registrar el usuario, y vincularlo al grupo que se haya especificado.

Para ello, se ha realizado un bucle *while* indicando que los separadores de cada línea están representados por comas, y almacenando cada campo en su variable correspondiente.

```
IFS=","
while read apellido1 apellido2 nombre email
do
…
```

Para utilizar como separador por defecto la coma, en *bash* se puede especificar a través del parámetro IFS, una variable interna especial.

Una vez obtenidos todos los campos para un alumno, es necesario generar un *nick* para poder registrarlo en BoloTweet.

Como el nombre no es único, y ya que el correo electrónico de la UCM sí lo es, lo que se ha hecho es utilizar el comando *cut* de nuevo para obtener la primera parte del correo, y utilizarla como nick.

```
nick=( $(echo $email | cut -d '@' -f1 | tr -d '.') )
```

El registro de un usuario precisa de una contraseña, por lo que se ha generado una temporal a partir del nick de usuario más un número aleatorio. Esta contraseña solamente se utiliza para el registro, ya que posteriormente, el usuario deberá modificarla por la que desee. Además, de esta manera, ni siquiera el administrador del sistema conoce nunca la contraseña de los usuarios.

```
number=( $(echo $RANDOM) )

pass=$nick$number
```

El nombre completo del alumno se ha generado a partir de los campos obtenidos del fichero CSV, y de una variable "*sp*" que contiene un espacio.

```
nameTemp=$nombre$sp$apellido1$sp$apellido2
```

Con todo esto ya se tiene lo necesario para registrar al usuario, por lo que solamente queda llamar al script PHP *registerUser.php* con los siguientes parámetros:

```
php registeruser.php -n$nick -w$pass -f$fullname -e$email
```

Como paso final, es necesario vincular a los alumnos con el grupo que corresponda. Sin embargo, para ofrecer una mayor funcionalidad en el script, también se puede ejecutar sin grupo, únicamente con lista de usuarios.

Para controlar esto, se ha realizado una comprobación de manera que si el parámetro que corresponde al *nickname* del grupo es distinto de vacío, se vinculen los usuarios al grupo, mientras que si no lo es, simplemente se den de alta.

```
if [ -n "$2" ];  then
php joingroup.php -n$nick -g$2
fi
```

Con todo esto, quedaría completamente funcional el script *registrarListas.sh*, capaz de registrar y vincular a un grupo existente una lista de alumnos de manera automática.

Como un alumno puede estar en más de un grupo, el script está realizado de manera que no falle aunque un usuario ya esté registrado, o ya pertenezca al grupo especificado.

La sintaxis de ejecución se especifica y se explica en el propio script

```
#!/bin/bash

# Este script está hecho para registrar listas de usuarios
# e insertarlos en sus respectivos grupos.
#
# Se deben respetar la siguiente estructura:
# El archivo csv tiene que tener 4 columnas, con el siguiente orden:
# -- apellido1 -- apellido2 -- nombre -- email
#
# El script se ejecuta de la siguiente forma:
#       ./registrarListas nombreCSV nickGrupo
#
# Siendo nombreCSV el nombre del fichero con la lista de usuarios.
# Siendo nickGrupo el nick exacto del grupo en Bolotweet.
```

Escalabilidad del perfil de los grupos

Ficheros modificados: groupaction.php, profilelist.php, grouplist.php, neo/css/ display.css

Uno de los aspectos a mejorar en BoloTweet, ha sido el comportamiento del sistema frente a grandes cantidades de usuarios, mensajes, grupos…

Por ello, ha sido necesario realizar algunos cambios en cuanto a la escalabilidad del *aside* derecho en los perfiles de los grupos.

Por defecto, aunque un grupo tenga más de 8 miembros, en el *aside* únicamente se muestran 8 avatares, teniendo que acudir a la sección de "Miembros" para ver el resto.
Sin embargo, se requiere poder acceder directamente y de manera rápida a cualquier usuario a través de su avatar, permitiendo además una fácil identificación.

Para solucionar esto, se han barajado diversas opciones:

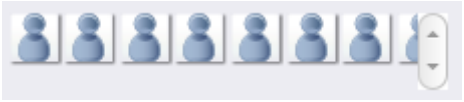
- La colocación de un botón "expandir" que permitiera ver todos los miembros.
- Añadir un enlace "mostrar todos", que llevará a una página con los miembros.
- Añadir una ventana emergente, un scroll, o similar.

Tras valorar todas las opciones, al final se ha decidido implementar la tercera. La opción de expandir puede estar bien, sin embargo, en el caso de que un grupo tenga 200 miembros, cuando se expanda va a ocupar demasiado espacio en el *aside*, quedando incómodo.

La opción del enlace "mostrar todos" es lo mismo que acceder a través del enlace de "Miembros", por lo que no tiene mucho sentido.

Por ello, se ha elegido la opción del *scroll*, por ser más cómoda y útil.

Para poder aplicar el scroll únicamente a la parte de los avatares, sin que se le aplique también al título de la sección, se ha tenido que modificar la función showMembers, de *groupaction.php*, para añadir un bloque *div*, al que posteriormente poder aplicar el scroll.



Sin embargo, tal y como se puede ver en la imagen, solamente se muestran 8 miembros, y además el *scroll* solapa el avatar del último usuario.

Por tanto, lo siguiente que se ha hecho es modificar esta restricción de que solamente se muestren 8 miembros, ya que ahora con el *scroll*, se deben mostrar todos.

En la función showMembers, en la llamada a getMembers, se ha quitado el último parámetro (que limita el número de usuarios a 27), de manera que se seleccionen todos los miembros del grupo.

```
$member = $this->group->getMembers(0);
```

A pesar de que en este punto ya se cargan todos los perfiles, se siguen mostrando únicamente 8. Para solucionar esto, en el fichero *profilelist.php*, se debe eliminar la siguiente comprobación:

```
$max = min($cnt, $this->maxProfiles());
```

Esta comprobación lo que hace es realizar un mínimo entre el número de perfiles del grupo, y la variable *maxProfiles*, que en este caso es 8. Como no se quiere límite de perfiles, esta comprobación no es necesaria.

Por otro lado, en la función showMembers del archivo *grouplist.php*, se realiza una comprobación que se encarga de generar un enlace "All Members" en caso que se hayan obtenido más de 27 miembros.

```
if ($cnt > MEMBERS_PER_SECTION)
```

Sin embargo, esta comprobación nunca sería cierta, ya que en la función getMembers se restringían los miembros a 27 (*bug interno de Statusnet*).

Con la modificación que se ha comentado anteriormente sobre la función getMembers, esta comprobación sí podría ser cierta, y por tanto el *bug* solucionado. No obstante, como en este caso se va a utilizar *scroll*, y se van a mostrar todos los miembros, no es necesario, por lo que se ha quitado toda la parte relacionada con el enlace "All Members".

Por último, solamente queda con reglas CSS dar un estilo correcto a ese bloque de miembros, para que no se corte el último usuario de cada fila, y para que el bloque siempre tenga un tamaño constante (4 filas de usuarios).
En el fichero *display.css* del tema neo, se ha modificado la anchura del bloque, para que quepan exactamente 7 miembros, y de esta manera no se solape el último miembro.

```
.section ul.entities { width: 220px; }
```

Por otro lado, se le ha asignado un *scroll* vertical al bloque, que solamente aparece si es necesario, y se ha ocultado *scroll* horizontal, ya que no es necesario. También se le aplica una altura máxima de 116px, que coincide exactamente con 4 filas de miembros.

```
.section #div-members-scroll{
```

```
                overflow-y: auto;
                overflow-x: hidden;
                max-height: 116px;
}
```

Todas estas reglas CSS se han añadido para el bloque "*Pending*", donde se muestran del mismo modo los avatares de los usuarios pendientes de unirse al grupo, y para el bloque "*Blocked*", donde se encuentran los avatares de los usuarios bloqueados.

Exceso de tamaño al registrar listas de usuarios

Ficheros modificados: Notice.php

En algunas pruebas del script "*registrarListas.sh*", se han detectado errores del siguiente tipo:

```
Ha habido un problema al guardar el mensaje. Es muy largo.
```

A pesar de que los usuarios se registraban correctamente, aparecían muchas advertencias similares a esta. Tras investigar de dónde procedía ese error, se comprobó que era ocasionado por los mensajes generados en el sistema del tipo:

alvaro se unió al grupo Sistemas web.

El texto en sí no excede el límite de caracteres, pero al ser generado con hipervínculos tanto en el usuario como en el grupo, sí que se sobrepasa el máximo permitido.

Esto es ocasionado porque todos los mensajes que aparecen en el sistema, ya sean anotaciones hechas por el usuario, o mensajes automáticos, son tratados como *notices*, y por tanto se les aplican sus mismas restricciones.

Este error estaba producido por el plugin *Activity*, encargado de lanzar mensajes automáticos con la actividad del sistema.

Sin embargo, un detalle interesante, es que si se unía a cualquier usuario desde la interfaz gráfica, en ningún momento se mostraba algún mensaje de error, y además sí que se publicaba el mensaje de actividad.

La explicación a esto es, que el plugin *Activity* está implementado de manera que realiza las mismas cosas, pero de formas diferentes, dependiendo desde dónde se le llame. Por esta razón, si se une a un usuario desde el script *joinuser.php*, sí se comprueba la longitud, mientras que si se realiza desde la interfaz, no.

Para solucionar esto, y evitar que futuros mensajes del plugin *Activity* puedan causar este tipo de errores, la solución ha sido añadir una excepción, de manera que a este tipo de "noticias" no se le controle la longitud.

Para ello, se ha estudiado el plugin *Activity*, para observar si existía algún tipo de atributo o valor que identificara a este tipo de mensajes, permitiendo así poder añadir una excepción.

Tras realizar diversas pruebas, se pudo observar cómo los objetos de tipo "notice", llevan un parámetro *source*, que les identifica respecto al resto de noticias.

En este caso, este parámetro viene representado por la constante

ActivityPlugin::SOURCE = "activity";

Como este parámetro va incluido en el objeto *notice*, que posteriormente se le pasa a la función saveNew, donde se realizan todo este tipo de comprobaciones, es tan sencillo como añadir una excepción, de manera que si el *source* del objeto recibido es del tipo "activity", no se realice la comprobación de límite de caracteres, y se eviten así este tipo de errores.

```
if($source!='activity'){
if (Notice::contentTooLong($final)) {
throw new ClientException(_('Problem saving notice. Too long.'));
      }
}
```

Desarrollar un Script para enviar correos de bienvenida

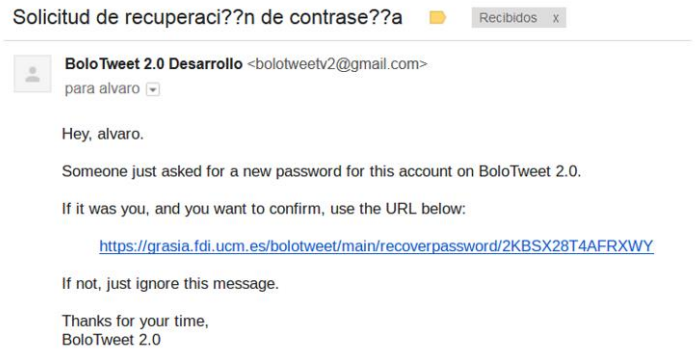
Ficheros modificados: User.php, mail.php, emailBienvenida.php

Uno de las necesidades en cuanto a la administración de BoloTweet, es la existencia de un mecanismo de envío de correos de bienvenida, a través de los cuales se les presente el sistema a los nuevos usuarios, y se les ofrezca una serie de indicaciones.

Como no existía ningún script que realizara esto exactamente, se ha tenido que desarrollar desde cero.

El primer problema que surge al plantearse esto, es si el sistema está preparado para enviar correos electrónicos a emails no confirmados. Para que un email sea válido, es necesario que el usuario lo verifique. En un principio, si esto provoca que no se puedan enviar correos a direcciones no verificadas, habría que modificar el comportamiento del sistema.

Por suerte, tras algunas pruebas con correos electrónicos no verificados, se vio como sí era posible recibir *emails*, por ejemplo, el de recuperación de contraseña.


Solicitud de recuperaci??n de contrase??a Recibidos x

BoloTweet 2.0 Desarrollo <bolotweetv2@gmail.com>
para alvaro

Hey, alvaro.

Someone just asked for a new password for this account on BoloTweet 2.0.

If it was you, and you want to confirm, use the URL below:

https://grasia.fdi.ucm.es/bolotweet/main/recoverpassword/2KBSX28T4AFRXWY

If not, just ignore this message.

Thanks for your time,
BoloTweet 2.0

Sin embargo, pueden observarse 2 grandes fallos. El primero, es que el contenido del mensaje no está traducido, y el segundo, que dependiendo del sistema gestor de correos, el título no se ve como se espera.

Por tanto, lo primero que se ha hecho es traducir el cuerpo del correo en la función *recoverPassword*, de User, y posteriormente buscar información sobre caracteres especiales en el *subject* de *emails*. Resulta que la codificación de los asuntos de los correos electrónicos, en este caso usan 7 bits, lo que provoca que los caracteres especiales no se muestren correctamente. De esta manera, la solución es indicar en el *subject* el tipo de codificación, y escribirlo en base64.

Como el *subject* es extensible a todos los correos, y para enviar un correo en el sistema siempre se hace a través de la función mail_to_user, se ha modificado la parte relativa al *subject* de los correos electrónicos, para que siempre se visualicen correctamente.

$headers['Subject'] = "=?UTF-8?B?".base64_encode($subject)."=?=";


Solicitud de recuperación de contraseña Recibidos x

BoloTweet 2.0 Desarrollo <bolotweetv2@gmail.com>
para alvaro

Hola, alvaro.

Alguien ha solicitado una nueva contraseña para esta cuenta en BoloTweet 2.0 Desarrollo.

Si has sido tú, y quieres modificarla, pulsa en el enlace de abajo:

https://grasia.fdi.ucm.es/bolotweetdevel/main/recoverpassword/84UR8N4Q6Q7A6U96MB6VHWW94N

Si no, puedes ignorar y eliminar este mensaje.

Gracias por tu tiempo,
BoloTweet 2.0 Desarrollo

Una vez corregido esto, se ha pasado a crear el script de bienvenida. A pesar de que existe un script para enviar correos electrónicos personalizados, *sendemail.php*, se ha desarrollado otro script independiente, ya que en este caso, ha sido necesario realizar una serie de pasos más complejos.

Para comenzar, se va a ofrecer la posibilidad de enviar el correo de bienvenida a un único alumno, a un grupo completo, o a todos los usuarios de BoloTweet.

Como este script se va a desarrollar en PHP, se han realizado en todo momento comprobaciones para controlar la existencia de los usuarios y de los grupos, de manera que no se puedan generar comportamientos inesperados.

Aprovechando que desde el script PHP se puede acceder a toda la información de la Base De Datos, se ha desarrollado una función dentro el script, que se encarga de generar un cuerpo de mensaje personalizado, y dedicado a cada usuario.

Otra de las mejoras incluidas en el script, es la generación de enlaces válidos de recuperación de contraseña. Cuando se registraban los usuarios a través del script *registerListas.sh*, se generaban contraseñas temporales, de manera que los usuarios deberían acudir a la sección de recuperación de contraseña, recibir un correo electrónico, y posteriormente modificarla.

Sin embargo, este comportamiento no es nada cómodo, por lo que tras estudiar el funcionamiento de recuperación de contraseña, se ha generado manualmente un enlace válido y único para cada usuario, de manera que al acceder, puedan modificar directamente la contraseña.

```
$confirm = new Confirm_address();
        $confirm->code = common_confirmation_code(128);
        $confirm->address_type = 'recover';
        $confirm->user_id = $user->id;
        $confirm->address   =   (!empty($user->email))   ?   $user->email   :
$confirm_email->address;

        if (!$confirm->insert()) {
            common_log_db_error($confirm, 'INSERT', __FILE__);
            throw     new     ServerException(_('Error     saving     address
confirmation.'));
return;        }
```

Como se puede ver, el código necesario para esta parte no es trivial, pues es necesario generar en todo momento códigos únicos y válidos, y almacenarlos en la base de datos.

Una vez definido el asunto del mensaje, realizadas las comprobaciones, y generado el cuerpo personalizado, se utiliza la función *mail_to_user* para enviar el correo electrónico al usuario.

Para la opción *--all*, que envía un correo electrónico a todos los usuarios de BoloTweet, se ha tenido que implementar una consulta SQL personalizada, ya que no existía ninguna función cuyo resultado fueran todos los usuarios del sistema.

```
$qry = "SELECT * FROM profile order by id asc";
```

Una vez obtenidos todos los perfiles, se van insertando en un array de miembros, para posteriormente, enviar un correo electrónico a cada uno de ellos.

El script *emailBienvenida.php* se puede encontrar en la carpeta *scripts*, y se puede obtener información acerca de su uso en el propio script. Además, se ha implementado de manera que muestre por pantalla el resultado de todas sus operaciones, pudiendo así controlar el proceso.

```
emailBienvenida.php [options]
Send a welcome email with instructions.

  -i --id       ID of the user to send email
  -n --nickname nickname of the user to send email
  -g --group    Nickname or alias of group to send email
  -G --group-id ID of group to send email
  -a --all      Send email to all members
```

Correo de bienvenida a los usuarios registrados

Ficheros modificados: registrarListas.sh

Cada vez que se registre un nuevo usuario, deberá recibir un correo de bienvenida. Para conseguir esto, se ha modificado el script *registrarListas.sh*, de manera que para cada usuario registrado, se llama al script *emailBienvenida.php*, consiguiendo así que reciba dicho correo.

Como es posible que al registrar listas de usuarios, alguno ya haya sido dado de alta a través de otra asignatura, es necesario controlar el registro, evitando así enviar dobles correos.

Se ha incluido por tanto una comprobación en el script, de manera que si el registro ha sido correcto, se envíe el *email* de bienvenida, mientras que si el usuario ya estaba registrado, se omita esta parte.

Como el script *registrarListas.sh* está desarrollado en *bash*, se ha utilizado la variable de retorno de comandos, *$?*.
Como el script *registerUser.php* devuelve el valor 0 si se ha registrado el usuario, y 1 si se produce algún error, se ha desarrollado la siguiente condición:

```
if [ $? -eq 0 ]; then
php emailBienvenida.php -n$nick
fi;
```

Modificado:

Esta manera de enviar correos de bienvenida se ha quitado de este script, y se ha colocado en el script *registerUser.php*.

La explicación completa se puede ver en la tarea *Envío manual de correo de bienvenida utilizando registerUser.php*.

Limpieza servidor de Desarrollo

Ficheros modificados: <Ninguno>

Al clonar el servidor de Producción para generar un servidor de Desarrollo, y ya que la carpeta raíz del proyecto tiene diferente nombre, por lo que se accede a través de una URL distinta, todo el contenido existente en este servidor, y almacenado en la base de datos, sigue apuntando a Producción.

Es similar a lo realizado en la tarea "*Problema con algunas URL en GRASIA, apuntan a 127.0.0.1*", con la diferencia de que en este caso no es posible borrar por completo la base de datos, ya que hay profesores y grupos que es conveniente preservar.

Para solucionarlo, lo primero que se ha hecho es eliminar todos los usuarios de Desarrollo, excepto los profesores, utilizando este mini-script bash:

```
for ((c=29; c<=123; c++))
do
php deleteuser.php -i$c -y
done
```

El siguiente paso ha sido modificar las URL de los profesores y los grupos que apuntan a producción. Para esto, StatusNet dispone de un script denominado *updateUrls.php*, que se encarga de actualizar las url almacenadas en el sistema.

Sin embargo, este script estaba mal desarrollado, y únicamente modificaba la URL del primer usuario. Se ha tenido por tanto que solucionar, para que corrija todas las URL de todos los usuarios. (Explicación en detalle en *bug internos*)

Tras corregir el script y ejecutarlo, las URI almacenadas en la tabla Profile, se han corregido correctamente de manera automática, pero las URIs de la tabla User, y Avatar, no es posible corregirlas de manera automática, ya que no se dispone de la información necesaria para recrear las direcciones automáticamente.

Por esta razón, se han tenido que modificar manualmente desde MySQL.

También se han actualizado rutas incorrectas que estaban almacenadas en la tabla config de la base de datos, y que se referían al servidor de producción.

- avatar:dir
- avatar:path
- site:locale_path
- site:path

Se ha borrado el contenido de la tablas deleted_notice, notice, notice_tag, conversation, fave, file, file_redirection, grades, group_inbox, inbox y reply.

Con todo esto, ha sido posible mantener el servidor de desarrollo sin tener que borrarlo por completo, y manteniendo así grupos y profesores.

Alumnos no deben poder eliminar sus anotaciones

Ficheros modificados: noticelistitem.php

Con el nuevo concepto de BoloTweet, con profesores, tareas, calificaciones… los alumnos no deben poder borrar sus mensajes.

Para solucionar esto, se ha valorado la mejor forma de realizarlo, y se ha considerado como mejor opción la creación de un nuevo rango, cuyo privilegio sea la posibilidad de borrar sus propios mensajes. De esta manera, únicamente los profesores, o personas muy particulares, podrán disponer de este rango, y serán capaces de eliminar sus anotaciones.

Esta opción ha sido la que se ha estimado más conveniente, ya que podría haberse realizado directamente utilizando el rango "*grader*", pero puede que sea necesario que algún usuario pueda borrar sus mensajes sin necesidad de ser profesor.

Para implementar esto, lo primero que se ha hecho es localizar qué archivo era el encargado de mostrar la opción de eliminar un mensaje, en este caso, *noticelistitem.php*.

```
($todel->profile_id == $user->i || $user->hasRight(Right::DELETEOTHERSNOTICE)))
```

Con esta comprobación, se podía borrar un mensaje siempre y cuando fuera del usuario actual, o bien se tuviera el derecho de borrar mensajes de otros.

Actualmente, para lograr el comportamiento que se requiere en Bolotweet, ha sido necesario añadir una nueva condición:

```
(($todel->profile_id == $user->id && $user->hasRole('deleter')) || $user->hasRight(Right::DELETEOTHERSNOTICE)))
```

De esta manera, únicamente si el mensaje es del usuario actual, y además ese usuario tiene el rango *deleter*, o bien si se tiene el derecho de borrar mensajes de otros, se podrá borrar el mensaje.

Creación de rango para borrar cualquier mensaje

Ficheros modificados: noticelistitem.php

Si en algún momento algún usuario enviara mensajes indebidos, es conveniente que alguien tenga la autoridad suficiente como para eliminar este tipo de mensajes.

Para solucionar esto, y utilizando la idea de la tarea anterior, se ha creado un nuevo rol "*maxDeleter*", que proporciona la posibilidad de borrar cualquier tweet de cualquier usuario.

Para asignar este rol a un usuario es preciso utilizar el script *userrole.php* de la siguiente manera:

```
php userrole.php -nnickUser –rmaxDeleter
```

Por último, de nuevo en el archivo *lib/noticelistitem.php*, se debe añadir una nueva condición para contemplar la posibilidad de ser *maxDeleter*.

```
(($todel->profile_id == $user->id && $user->hasRole('deleter')) || $user->hasRole('maxDeleter') || $user->hasRight(Right::DELETEOTHERSNOTICE)) { … }
```

Crear script para realizar copias de seguridad

Ficheros modificados: backupBT.sh

Al utilizar un método de trabajo basado en desarrollo y producción, es necesario que se realicen copias de seguridad antes de realizar cualquier cambio.

Hasta el momento, y debido a la inexistencia de herramientas para la administración del sistema, este *backup* se realizaba generando una copia completa de la MV donde está corriendo BoloTweet. Sin embargo, este proceso es demasiado lento y demasiado pesado, ya que es necesario clonar unos 3gb de MV, cuando realmente el sistema completo no ocupa más de 50 MBytes, por lo que es necesario agilizar y automatizar este proceso.

Por ello, la siguiente decisión que se ha tomado es decidir en qué lenguaje desarrollar el script. Sin embargo, ya que va a ser necesario realizar bastantes acciones sobre el sistema a nivel de sistema operativo, tales como copia y borrado de ficheros, compresión de archivos, acceso a la base de datos… se ha preferido desarrollar en *bash.*
La idea fundamental del script es realizar una copia de seguridad de la carpeta completa de la web, localizada en /var/www/, y de la base de datos.

Este script es crítico, ya que una mala instrucción puede dejar inaccesible por completo el sistema. Por ello, se ha desarrollado completamente en local, y se han realizado exhaustivas pruebas hasta comprobar su funcionamiento.

El siguiente paso ha sido decidir en qué formato y con qué estructura se quería realizar el fichero de BackUp.

Tras comparar varias alternativas, la decisión final ha sido generar un fichero comprimido cuyo nombre será la fecha del día en que se realizó el backup, y cuyo contenido será una carpeta comprimida con las fuentes de BoloTweet, junto con un archivo *.sql con una copia de la base de datos.

Un detalle importante del script, es que se han ido implementando comentarios acerca de las salidas de los comandos, para poder visualizar por completo el proceso.

Una vez decidido todo lo necesario para la realización del script, a continuación se comentarán todos los pasos necesarios para su implementación:

1. Para evitar que durante el BackUp se pueda corromper la base de datos, al comienzo de la ejecución se parará el servidor Apache, dejando momentáneamente la página web inaccesible.

```
sudo -p "Introduce tu contraseña: " service apache2 stop > /dev/null 2>&1
```

2. El servicio MySQL no es necesario pararlo por dos razones. La primera es que con el servidor Apache caído no se puede acceder a la base de datos, y la segunda, que va a ser necesario que esté activo para realizar la exportación de los datos.

3. Desde este punto y a lo largo de todo el script, se han implementado comprobaciones utilizando la variable de retorno bash, para poder controlar la ejecución de los comandos.

```
if [ $? -eq 0 ]; then
printf "Listo!\n"
else
printf "Fallo. Abortando...\n"
exit 1
fi
```

4. Para generar un archivo comprimido con la fecha actual, se ha tenido que localizar una función *bash* que devolviera una salida apropiada. La función escogida ha sido *date*.

```
timeStamp=$(date '+%F')
```

5. El nombre del fichero comprimido con las fuentes de BoloTweet, y del fichero *.sql son siempre los mismos, por lo que se han definido manualmente en el script.

6. Para la copia de seguridad de la carpeta con todas las fuentes, se ha utilizado el comando *tar* con las siguientes opciones:

```
tar -Pzcf ~/$nameFile --exclude='.git' www/
```

7. Si esta copia ha sido satisfactoria, se procede al respaldo de la base de datos MySQL. Para desarrollar este paso, se han barajado bastantes opciones, ya que existen diferentes maneras de realizar una copia de seguridad.

   Finalmente se ha utilizado *mysqldump*, cuyo resultado es un fichero *.sql con las consultas y los datos necesarios para recrear la base de datos actual.

```
mysqldump -u adminbtdb -p bolotweetdb > ~/$nameSql
```

8. Si este paso ha sido correcto, y por tanto se dispone de un archivo comprimido con las fuentes, y un fichero *.sql con la base de datos, únicamente falta comprimirlos en un archivo final con la fecha del día.

   Para ello, de nuevo se ha utilizado el comando *tar* de la siguiente forma:

```
tar -Pzcf ~/$nameFolder {"$nameFile","$nameSql"}
```

9. Si no se ha producido ningún error, solamente queda borrar los archivos temporales, es decir, el primer archivo comprimido, y el fichero *.sql.

```
rm ~/$nameFile ~/$nameSql
```

10. A continuación se levanta de nuevo el servidor Apache

```
sudo service apache2 start > /dev/null 2>&1
```

Con esto quedaría terminado el script *backupBT.sh*, el cual se puede observar detenidamente en la carpeta */scripts*.

No obstante, algunos detalles a comentar. El primero es que se han controlado en todo momento la salida de los comandos, para que no interfirieran con los mensajes de salida incluidos en el script, redirigiéndolos a */dev/null* para que no se muestren.

Otro detalle importante es que el script solicitará la contraseña *root* del usuario para poder parar y arrancar Apache, y la contraseña del usuario de la Base de Datos, para poder acceder a la misma y realizar el respaldo.

El último detalle a comentar, es que el script está preparado para ser ejecutado desde cualquier lugar del sistema, pues en todo momento trabaja con rutas absolutas, y no hay riesgo de un comportamiento inesperado.

```
alvaro@alvaro-linux:/var/www/bolotweet/scripts$ ./backupBT.sh
Este script necesita ejecutar algunos comandos como SuperUsuario.
Presiona [Enter] para empezar el BackUp...
Introduce tu contraseña:
Parando Apache...Listo!
Creando Backup de /var/www/...Listo!
Creando Backup de la Base de Datos...Listo!
Creando Fichero Final...Listo!
Borrando archivos temporales...Listo!
Iniciando Apache...
Listo!
BackUp terminado

Presiona [Enter] para terminar...
```

Crear script para restaurar la copia de seguridad

Ficheros modificados: restoreBT.sh

Como es normal, para completar aún más las tareas de administración del sistema, y ya que se ha desarrollado un script para realizar un *backup* del sistema, es conveniente desarrollar otro que sea capaz de restaurar de manera automática cualquier copia de seguridad desarrollada por el script anterior.

El desarrollo de este script es bastante más complejo y peligroso, pues se va a reemplazar por completo tanto la base de datos, como la carpeta con la web, con el riesgo que esto conlleva.

Para conseguir seguridad, de nuevo se han implementado comprobaciones de todas las acciones a través de la variable de retorno de bash ($?).

Para ejecutar este script, es necesario indicar como parámetro el nombre del fichero con la copia de seguridad de BoloTweet.

El fichero de *backup* debe haber sido creado con el anterior script, y tener el formato esperado, ya que de lo contrario el proceso se abortará.

A continuación se comentarán los pasos necesarios para la implementación del script:

1. En primer lugar, de la misma manera que en el script de *backup*, se ha parado el servidor Apache para evitar comportamientos inesperados sobre el sistema.

2. Se definen los nombres de archivo predeterminados, necesarios para poder extraer posteriormente los ficheros.

3. Previa comprobación de si el fichero pasado como parámetro existe, se extrae todo su contenido.

```
if [ -e "$1" ]; then
tar -xzf $1
```

4. Si este proceso se produce sin errores, se tendría por un lado un fichero comprimido con las fuentes de BoloTweet, y por otro lado el archivo *.sql con la base de datos. Tras comprobar que ambos ficheros existen, se procede a extraer de nuevo el segundo archivo comprimido:

```
if [ -e "$nameFile" ]; then
tar -xzf $nameFile
```

5. A continuación se comenzará a restaurar la copia de seguridad. Para ello, lo primero es eliminar el contenido de la carpeta */var/www/*.

```
sudo rm -r /var/www/* > /dev/null 2>&1
```

6. Si se ha borrado correctamente, se procede a copiar el nuevo contenido.

```
sudo cp -R www/* /var/www/
```

7. El siguiente paso es restaurar la base de datos MySQL. Para ello, es necesario que la base de datos exista y esté vacía, por lo que el siguiente paso es eliminarla si es necesario:

```
mysqladmin -uadminbtdb -p -f drop bolotweetdb 2>/dev/null
```

8. A continuación se procederá a crear una base de datos nueva y vacía.

```
mysqladmin -u "adminbtdb" --password create bolotweetdb
```

9. Una vez creada la base de datos, es necesario importar el contenido del fichero *.sql. Existen diversas formas de importarlo, utilizando mysql, php, mysqladmin… pero se ha optado por utilizar mysql.

```
mysql -u uadminbtdb -p bolotweetdb < $nameSql
```

   Este proceso es posible que se demore un poco, dependiendo de la cantidad de datos existentes en la base de datos a restaurar.

10. Por último, si todo ha ido bien, se procede a borrar los archivos temporales, es decir, los ficheros extraídos para la restauración.

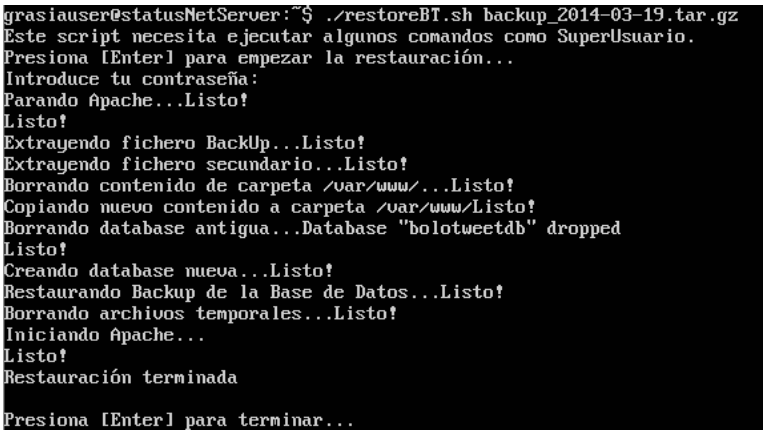
```
rm -R www/ bbdd.sql web_files.tar.gz
```

11. Si todo el proceso se ha producido sin errores, se arrancará de nuevo el servidor Apache, y la página debería estar de nuevo activa y restaurada por completo.

Con todos estos pasos, se ha conseguido desarrollar el script *restoreBT.sh*, encargado de automatizar por completo la restauración del sistema BoloTweet.

Por último, comentar algunos detalles a tener en cuenta. Al comienzo de la ejecución se solicitará la contraseña *root*, para poder trabajar sobre el servidor Apache, y para trabajar sobre el directorio /var/www. Por otro lado, mencionar también que se deberá proporcionar la contraseña del usuario de la base de datos MySQL, ya que es necesario para poder trabajar sobre la base de datos.

De la misma manera que en el script backupBT.sh, se han controlado todas las salidas de los comandos, de manera que durante la ejecución de la restauración, se puede ir observando en todo momento qué se está realizando.



Events y Polls solamente disponible para profesores

Ficheros modificados: microappplugin.php

Tal y como está planteado BoloTweet, los eventos y las encuestas únicamente deben ser creados por profesores, por lo que se ha tenido que restringir la creación de este tipo de objetos únicamente a usuarios con rango *grader*.

Para conseguir esto, y ya que ambos objetos son creados a través de Plugins, se ha tenido que estudiar el funcionamiento de los mismos, para conocer dónde y de qué manera se puede añadir esta restricción.

El proceso ha sido bastante más complejo de lo que se esperaba, ya que en los propios Plugins, no se controla la creación de los botones, que posteriormente servirán para crear encuestas y eventos.

Estos dos plugins, al igual que Bookmark y Question, tienen un formato especial, y extienden de la clase *microappPlugin*, que en este caso es quien controla la creación de todos sus hijos.

Esto quiere decir que los plugins no tienen ningún evento, ni utilizan ningún hook para mostrar el botón, si no que sobrescriben métodos de microappplugin, y es esta última la que usa el evento *onStartShowEntryForms* para mostrarlos.

Por tanto, la única opción para controlar esto, se reduce a añadir las comprobaciones en la clase microappPlugin.

En el método onStartShowEntryForms, por defecto lo único que se hacía era lo siguiente:

```
$tabs[$this->tag()] = $this->appTitle();
```

Como este hook es sobrescrito por todos los plugins anteriormente mencionados, en especial Event y Poll, por este método pasan todos los plugins en su creación.

Por ello, es ahí donde debe controlarse qué se debe mostrar. El siguiente paso ha sido depurar la ejecución, para observar los posibles valores de *appTitle* y *tag*, y poder así realizar comprobaciones.

Una vez conocidos estos valores, se han añadido dos condiciones, de manera que si el *tag* recibido coincide con 'event', o con 'poll', solamente se añada al *array $tabs* si el usuario actual tiene el rango *grader*.

La función por tanto ha quedado de la siguiente manera:

```
function onStartShowEntryForms(&$tabs) {

        if ($this->tag() == "event" || $this->tag() == "poll") {

            $user = common_current_user();

            if ($user->hasRole('grader')) {

                $tabs[$this->tag()] = $this->appTitle();
            }
        } else {

            $tabs[$this->tag()] = $this->appTitle();
}

        return true;
    }
```

Con estas condiciones, ha sido posible restringir estas dos opciones únicamente a profesores, consiguiendo además un sistema de exclusiones realmente sencillo, ya que si se quisiera restringir algún otro *plugin*, tan sólo debe añadirse una condición más.

Nombres de los grupos de la columna de la izquierda se ven muy juntos y mal

Ficheros modificados: groupsnav.php, base/css/display.css

Por defecto en el *aside* izquierdo, se muestran los grupos a los que el usuario actual pertenece. Sin embargo, se muestran los nombres completos, de manera que cuando se pertenece a más de un grupo, el resultado es confuso, ya que no se distinguen fácilmente.

La idea para solucionar esto es reemplazar los nombres completos por sus abreviaturas (*nicknames*), de manera que se pueda ver rápidamente el grupo que corresponde, y se diferencien con facilidad.

Para corregir esto, se ha tenido que modificar la función getItems del fichero */lib/groupsnav.php*.

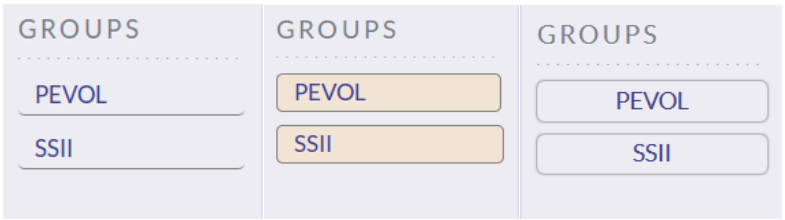
Esta función se encarga de obtener los grupos a los que pertenece el usuario actual, y almacenarlos en un *array*. Cuando los almacena, en el campo que corresponde con la etiqueta, por defecto se almacena el mejor nombre para el grupo, es decir, el *nickname*, o bien el *fullname* si lo tiene.

En este caso se desea que siempre almacene el *nickname*, por lo que es tan sencillo como modificar este valor en la creación del *array*, quedando de la siguiente manera:

```
function getItems()
    {
        $items = array();

        while ($this->groups->fetch()) {
            $items[] = array('placeholder',
                             array('nickname' => $this->groups->nickname,
                                   'mainpage' => $this->groups->homeUrl()),
                             $this->groups->nickname, // LABEL
                             $this->groups->getBestName()
                            );
        }

        return $items;
}
```

Por último, se ha querido modificar ligeramente a través de reglas CSS el estilo de este apartado, para lograr una mejor y más rápida diferenciación de los grupos.

Tras barajar diferentes alternativas:



Se ha decidido mantener la tercera, ya que es la más clara y la que menos rompe la estética de la página.

Se ha modificado el fichero */theme/base/css/display.css*, y se han añadido las siguientes reglas

```
#site_nav_local_views ul#nav_groups a{

      box-shadow: 0px 0px 1px rgb(66, 66, 66);
      text-align: center;
}
```

Desarrollar un script para crear grupos

Ficheros modificados: createGroup.php

Actualmente, StatusNet no dispone por defecto de ningún script para crear grupos, una acción que sí debe poder ser realizada desde línea de comandos, y que además es necesaria para cumplimentar la parte de administración del sistema.

Sin embargo, hay que tener en cuenta algunos detalles antes de desarrollar el script, pues un grupo ofrece diferentes tipos de configuración. En primer lugar, hay que valorar si el grupo se debe crear como Privado o Público, ya que influirá por completo en la actividad del mismo, y en la manera en que los usuarios podrán unirse a él.

Además, cuando se crea un grupo de manera gráfica, el usuario que lo crea pasa a ser directamente el administrador del grupo, y quien podrá posteriormente moderarlo a su antojo.

En este caso, como el grupo se crea a través de un script, va a ser necesario indicarle qué usuario es el que se quiere vincular como administrador.

Por todo ello, este script necesita como parámetro el *nickname* del profesor que quiere crear el grupo, y que además se convertirá en el administrador del grupo, y por otro lado el *nickname* escogido para la asignatura.

En cuanto a la implementación del script, realizado en PHP, se han realizado comprobaciones para controlar que el usuario pasado como parámetro exista, antes de pasar a la creación del grupo.

Posteriormente, se ha comprobado que el *nickname* del grupo sea válido, que no se encuentre en la lista negra del sistema, y que no exista un grupo cuyo *nickname* o alguno de sus alias coincidan con el *nickname* proporcionado.

Antes de pasar a la creación del grupo, es necesario establecer la visibilidad del mismo. En este caso, se ha decidido que se cree como privado, pudiendo posteriormente el profesor modificarlo a su antojo.

```
$force_scope = 1;
$join_policy = User_group::JOIN_POLICY_MODERATE;
```

Si todo va bien, se hace uso de la función *register* de *User_Group*, la cual se encarga de todo el proceso de creación y almacenamiento del grupo.

```
User_group::register(array('nickname' => $nickname,
    'userid'   => $profile->id,
    'join_policy' => $join_policy,
    'force_scope' => $force_scope,
    'local' => true));
```

Como solamente se desea registrar el nuevo grupo, y ya que el profesor podrá posteriormente completar toda la información que desee acerca del mismo, únicamente se le proporciona a la función *register* lo imprescindible:

- Nick del grupo
- Force Scope -> El grupo se crea como privado
- Join_Policy -> El grupo se crea como privado
- Local -> Ámbito del grupo
- userid -> Creador y Administrador del grupo

Con todo esto queda desarrollado el script *createGroup.php*, permitiendo así la creación de cualquier grupo a través de línea de comandos.

En el propio script está definido el uso del mismo:

```
createGroup.php [options]
Create a group and admin.

  -n --nickname nickname of the user to vincule (admin)
  -g --group    Nickname or alias of group
```

Problema al vincular usuarios a grupos privados

Ficheros modificados: User.php, Profile.php, joinGroup.php

Por defecto, cuando un grupo es privado en StatusNet, significa que, por un lado, los mensajes del grupo no son visibles al resto de usuarios, y por otro, que los usuarios no se pueden unir directamente al grupo, si no que solicitarán una petición de unión al mismo.

Este comportamiento es el esperado cuando se realiza desde la interfaz, sin embargo, si se realiza a través de scripts, se produce un comportamiento inapropiado.

Cuando el administrador del sistema registre una nueva asignatura, por defecto se hará con visibilidad privada. Al insertar por tanto usuarios en esa asignatura, todos quedarán pendientes de aprobación, teniendo el profesor del grupo que aprobarlos uno por uno.

Por ello, es necesario modificar el comportamiento del sistema en el caso de que sea un script quien realice la unión a los grupos.

Se han barajado varias opciones para solucionar esto, entre ellas la posibilidad de cambiar la visibilidad del grupo a público antes de vincular usuarios, y la modificación posterior una vez finalizada esta vinculación.

Sin embargo, esto no es una buena solución, entre otras cosas porque durante ese tiempo se está dejando ese grupo con visibilidad pública.

Por todo ello, la solución que se ha realizado ha sido modificar el funcionamiento interno de la función joinGroup, de User, así como del script *joinGroup.php*.

Ahora, la función joinGroup recibe un nuevo parámetro (script), que propagará a la función joinGroup de Profile, también modificada para recibir el nuevo parámetro.

Cuando se llame a la función joinGroup desde el script, se pasará un valor *true* para ese nuevo parámetro. A su vez, se ha añadido una condición nueva a la comprobación realizada en la función *joinGroup* de *Profile*, de manera que si el parámetro script es true, los usuarios se vinculen directamente al grupo, independientemente de la visibilidad del mismo.

```
if(($group->join_policy == User_group::JOIN_POLICY_MODERATE) && !$script) { … }
```

Añadir información de salida a algunos scripts

Ficheros modificados: createGroup.php, leaveGroup.php, joinGroup.php, resgisterUser.php, makeGroupAdmin.php.

Algunos scripts incluidos en *BoloTweet*, no incluyen ningún tipo de información acerca de las acciones que realizan. Por ello, se han modificado los siguientes scripts añadiendo comprobaciones e información en todo momento sobre el estado del proceso:

- *createGroup.php*
- *leaveGroup.php*
- *makeGroupAdmin.php*
- *joinGroup.php*
- *registerUser.php*

Envío manual de correo de bienvenida utilizando registerUser.php

Ficheros modificados: registerUser.php, registrarListas.sh

Cuando se registran usuarios a través del script *registrarListas.sh*, se implementó que a cada nuevo usuario registrado se le enviara un correo con indicaciones utilizando el script *emailBienvenida.php*. (Explicación con detalle en tarea *Correo de bienvenida a los usuarios registrados)*

Sin embargo, si se registra un usuario con el script *registerUser.php*, es el administrador quien debe enviar el correo de bienvenida.

Esto no es el comportamiento esperado, por lo que se ha modificado de manera que ahora se envía un correo electrónico desde *registerUser.php*. Como el script *registrarListas.sh* a su vez llama a este script para registrar a los usuarios, se les va a enviar igualmente un correo de bienvenida.

Como no es sencillo ni fiable llamar a un script PHP desde otro (se podría hacer con el comando *exec*, pero no siempre funciona), lo que se ha hecho es adaptar el contenido del script *emailBienvenida.php*, e insertarlo en el script *registerUser.php*.

De esta manera, tanto si se utiliza el script *registrarListas.sh* como *registerUser.php*, siempre se enviará automáticamente un correo electrónico de bienvenida con indicaciones.

Eliminar PHPSESSID de la URL

Ficheros modificados: .htaccess, php.ini, util.php

Las URL que se generan en BoloTweet en el servidor de GRASIA llevan incluido el parámetro *PHPSESSID*.

Este parámetro puede provocar problemas de seguridad, por lo que es importante que no se añada a las URL.

Para solucionarlo, se han realizado los siguientes pasos:

- Editar la configuración interna de PHP, a través del fichero *php.ini*, modificando los siguientes campos:

```
session.use_cookies=1
session.use_only_cookies=1
session.use_trans_sid=0
```

- *Modificar la función common_local_url* y *common_path* para fijar por defecto *addSession* a *false*.

- Reescribir la URL desde el fichero *.htaccess*, con las siguientes reglas:

```
RewriteCond %{QUERY_STRING} PHPSESSID=.*$
  RewriteRule .* %{REQUEST_URI}? [R=301,L]
```

Crear nuevos hooks onStartToolsLocalNav y onEndToolsLocalNav

Ficheros modificados: defaultlocalnav.php

Al crear el nuevo *plugin* de Tareas, y tener que añadir un nuevo enlace debajo del menú "Herramientas", ya no es posible utilizar el *hook* que se ha utilizado en *NotesPDF* para colocar el enlace de Apuntes, *startDefaultLocalNav,* pues no va a permitir crear un enlace dentro del menú creado en *NotesPDF*.

Para corregir esto, la solución ha sido crear la sección "Herramientas" de manera interna, en el *core* de *BoloTweet*, y posteriormente crear dos nuevos *hooks*, que permitan insertar elementos dentro de esta nueva sección.

Para ello, se ha modificado el fichero *lib/defaultlocalnav.php*, y antes de la sección "*Home*", se ha insertado el siguiente código:

```
if (!empty($user)) {
                $this->action->elementStart('li');
                $this->action->element('h3', null, 'Herramientas');
                $this->action->elementStart('ul', array('class' => 'nav'));
                Event::handle('StartToolsLocalNav', array($this));
                Event::handle('EndToolsLocalNav', array($this));
                $this->action->elementEnd('ul');
                $this->action->elementEnd('li');
}
```

Como se puede ver, se ha añadido el título de la sección, y los nuevos *hooks* se han creado dentro de la lista de elementos, de manera que ahora sí, desde fuera, sea posible insertar enlaces dentro de ese apartado de "*Herramientas*".

Cambiar el logo de StatusNet

Ficheros modificados: logo.png

Para añadir un logo personalizado, simplemente se ha tenido que diseñar un nuevo logo, redimensionarlo al tamaño del original, colocarlo en la carpeta del tema neo-blue, que es el que se está utilizando, y llamarlo de la misma manera que el anterior.

De esta manera, sin tocar nada más se sustituye el antiguo logo por el de BoloTweet.

Añadir logo versión móvil

Ficheros modificados: mobile-logo.png

Para sustituir el logo de la versión móvil, se ha tenido que realizar los mismos pasos que en la tarea anterior. Se ha diseñado el nuevo logo, se ha ajustado al tamaño del anterior, y se ha sustituido en la carpeta neo-blue por el antiguo, de manera que el resultado es el siguiente:

Cambiar tema por defecto

Ficheros modificados: <Ninguno>

Para modificar el tema actual, y a pesar de existir diferentes maneras de hacerlo, se ha decidido utilizar el script setconfig.php para cambiarlo a nivel de configuración del sistema.

```
$config['site']['theme'] = 'neo-blue';
```

Plugins

Grades

La idea de este plugin es, principalmente, ofrecer una herramienta de evaluación de anotaciones. Los profesores serán capaces de puntuar y calificar mensajes, pudiendo así ofrecer un *feedback* frente a las anotaciones de los alumnos.

Además, también se ofrecerá una opción para visualizar un listado con los alumnos de cada grupo, junto con sus puntuaciones totales, ofreciendo incluso la posibilidad de exportar estos listados a un fichero CSV.

Adaptación de Grades a StatusNet 1.1.1

Ficheros modificados: GradesPlugin.php, grades.css, Grades.php, grades.js, GradeForm.php, GradeAction.php, GradereportAction.php

- Cambiada por completo la estructura del *plugin*. Se ha separado y aislado del core de StatusNet.
- Organizados todos los ficheros en nuevas carpetas (actions, classes, css, js, scripts, lib).
- Modificada la manera de cargar e importar archivos, ahora se usa el *hook* apropiado *onAutoLoad*.
- Modificada la manera de enlazar CSS, ahora se utiliza el *hook* apropiado *onEndShowStyles*.
- *El menú GradeReports* cambiado al principio del *primary nav*.
- Función *onCheckSchema* cambiada por completo. Ahora cada clase tiene su propia función *schemaDef*, a la que se llama desde *onCheckSchema*.
- Cambiadas rutas absolutas de todos los archivos por la nueva localización.
- Seleccionadas reglas CSS que afectaban al *plugin*, y agrupadas en un archivo *grades.css* aislado del core, y vinculado únicamente con el plugin.
- Las clases que se relacionan con la base de datos se han cambiado a Managed_DataObject, en vez de Memcached_DataObject.
- La función *register* guardaba mal los datos, y con ID repetido. Solucionado cambiando el tipo del ID en la BBDD para que se autoincremente.
- Archivo *status.ini* eliminado, ya no se utiliza.
- Reglas CSS cambiadas por completo para dar una apariencia distinta a Grades.

Varias puntuaciones sobre un mismo tweet generan registros distintos

Ficheros modificados: GradeAction.php, Grades.php

Cuando se puntuaba un mensaje ya calificado, se almacenaba un nuevo registro en la base de datos asociado a esa noticia. Este comportamiento se ha modificado, de manera que una puntuación de un mismo profesor sobre una misma noticia, actualice la nota del registro existente.

Para solucionarlo, en el fichero *GradeAction.php*, cuando se recibe una puntuación, se comprueba si existe una noticia ya puntuada con el mismo *noticeID*. Si existe, se redirige a la función de la clase Grades *updateNotice*, modificándose únicamente la fecha y la puntuación, pero conservando el registro.

```
$exist = Grades::getNoticeGrade($noticeid, $nickname);

if ($exist != '?') {

            Grades::updateNotice(array('noticeid' => $noticeid,
'grade' => $gradevalue, 'userid' => $nickname));
        }
```

Para actualizar el registro existente, se ha realizado una consulta SQL, ya que el método update del que disponen todos los objetos Managed_DataObject no funciona correctamente.

```
$qry = 'UPDATE ' . $user_table .
                ' SET grade=' . $grade .
                ', cdate=\'' . $time . '\'' .
                ' WHERE noticeid=' . $noticeid .
                ' AND userid="' . $userid . '"';
```

Al puntuar mensajes aparece "DB Error: already exists"

Ficheros modificados: Grades.php

Con el comportamiento de Grades tras migrarlo a la nueva versión de StatusNet, cada vez que se puntuaba un mensaje y se insertaba un nuevo registro en la base de datos, su ID se generaba manualmente, utilizando el comando UUID::gen().

Sin embargo, este comando generaba salidas inesperadas, y en ocasiones generaba ID existentes, provocando que la inserción de un nuevo registro en la base de datos mostrar el error anteriormente comentado.

Para solucionar esto, lo que se ha hecho es modificar la estructura interna de la base de datos de Grades, de manera que la columna ID de la tabla Grades ahora es del tipo *serial* (big int con auto increment).

De esta manera, el ID es controlado en todo momento por MySQL, y se evitan cualquier serie de errores ocasionados por conflictos entre ID.

Corregido método getGroupsWithGrades

Ficheros modificados: Grades.php

Tal y como estaba desarrollado en la versión anterior, se seleccionaban los *tags* que aparecieran en algún mensaje puntuado, y que coincidieran con algún nombre de grupo.

Sin embargo, lo que se espera de esta función es que se seleccionen los *grupos* con puntuaciones,

independientemente de las etiquetas (*tags*).
Para ello, se ha modificado por completo la consulta SQL, de manera que ahora se logra el comportamiento esperado:

```
$qry = 'SELECT gi.group_id as groupsIDs' .
                ' FROM grades, group_inbox gi WHERE ' .
                ' gi.notice_id = grades.noticeid' .
                ' group by gi.group_id';
```

Corregido método getGradedNoticesAndUsersWithinGroup

Ficheros modificados: Grades.php

En la versión anterior de Grades, esta función lo que hacía era obtener un listado de usuarios que tienen mensajes puntuados con etiquetas con el nombre de un determinado grupo.

Sin embargo, esta función lo que debe devolver es el listado de usuarios junto con su puntuación para un grupo dado, independientemente de las etiquetas.

Se ha solucionado modificando de nuevo la *query* por completo.

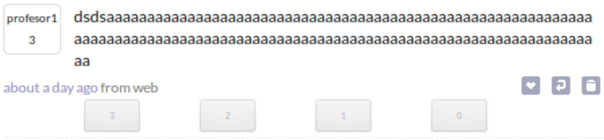
```
$qry = 'select p.nickname as userid, sum(g.grade) as grade' .
                '  from  grades  g,  group_inbox  gr,  notice  n,  profile  p,
local_group lg' .
                ' where g.noticeid = gr.notice_id' .
                ' and lg.nickname = \'' . $groupnick . '\'' .
                ' and gr.group_id = lg.group_id ' .
                ' and g.noticeid = n.id ' .
                ' and n.profile_id = p.id' .
                ' group by p.nickname';
```

En la opción "Perfil" de cualquier usuario las puntuaciones no se ven correctamente

Ficheros modificados: grades.css

Como en el modo Perfil no se muestran los avatares del usuario en cada mensaje, el recuadro con la puntuación no se ve, ni está colocado correctamente.

Para solucionarlo, y tras probar distintas alternativas, se han añadido una serie de reglas en el fichero grades.css, de manera que la puntuación *flote* a la izquierda del texto del mensaje.



<u>Eliminar botones de puntuación en tweets puntuados</u>

Ficheros modificados: GradesPlugin.php

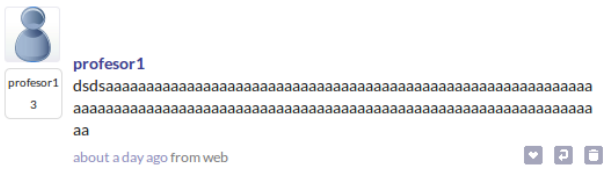
Tal y como estaba implementado en la primera versión de Grades, siempre que un tweet podía ser puntuado por el usuario actual, se mostraban los botones para calificarlo. Sin embargo, el comportamiento deseado es que cuando un mensaje ya haya sido calificado, no se deben mostrar estos botones.

Para solucionarlo, en la función *onEndShowNoticeItem* de *GradesPlugin.php*, antes de mostrar el bloque con los botones, se ha añadido una comprobación, de manera que si ese tweet ya había sido puntuado, no se muestre el bloque para puntuar.

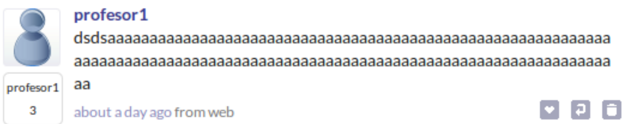
```
$gradevalue = Grades::getNoticeGrade($noticeid, $nickname);

if ($gradevalue != '?') { … }
```

<u>Tweets puntuados descuadran el contenido del mensaje</u>

Ficheros modificados: grades.css



Para solucionarlo, se ha modificado el valor de la propiedad float por left, de manera que ahora se autoajusta siempre al contenido del tweet, tenga 1 o varías líneas.



Sin embargo, al realizar esto, en los tweets que tienen una sola línea la puntuación se solapa con el mensaje de abajo, por lo que se ha tenido que añadir la siguiente regla CSS para solucionarlo:

```
margin-bottom: 35px.
```

Tweets sin puntuar descuadran el avatar

Ficheros modificados: GradesPlugin.php

En la versión anterior de Grades, siempre se cargaba un bloque *div* para la puntuación en los mensajes, estuvieran o no puntuados.

Esto provocaba que en los mensajes sin puntuar, se descuadrara por completo el avatar. Como no es necesario generar un recuadro para los mensajes no puntuados, para solucionarlo lo que se ha hecho es rechazar este uso, de manera que únicamente se reserva hueco para una puntuación si un mensaje está puntuado.

Los tweets de profesores también pueden ser puntuados

Ficheros modificados: GradesPlugin.php

Tal y como estaba implementada la versión anterior, cualquier tweet podía ser puntuado, fuera de quien fuera. Este comportamiento no es correcto en la versión actual, por lo que los tweets escritos por profesores no pueden ni deben ser puntuados.

Por ello, se ha añadido una nueva condición en la función *onEndShowNoticeItem*, de manera que si el autor de la noticia tiene el rango de *grader*, no se muestren los botones, aunque el usuario actual sea otro *grader*.

```
if (!$args->notice->getProfile()->getUser()->hasRole('grader'))
```

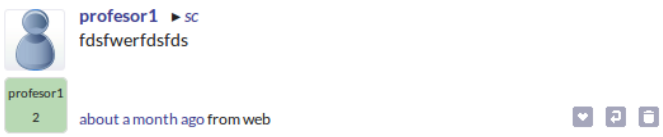
El color de las puntuaciones no se ve fácilmente

Ficheros modificados: grades.css

Se ha modificado el color de fondo del recuadro con la puntuación, de manera que se puedan ver y distinguir del resto del mensaje más fácilmente.

Se ha añadido una nueva regla CSS en su fichero correspondiente (grades.css).

```
background-color: #B8D9B4;
```



Botón para modificar las puntuaciones

Ficheros modificados: GradesPlugin.php, grades.js, grades.css

Lo primero que se ha hecho es localizar qué *hook* permite añadir un nuevo elemento al lado del resto de botones de opciones del *tweet*.

Una vez localizado el hook, *StartShowNoticeOptionItems*, es importante conocer cuál es la mejor forma de ofrecer este sistema de actualización de puntuaciones.

En un primer momento se intentó a través de un *form* con una acción asociada, pero finalmente se ha optado por un sistema completamente diferente.

La idea es que siempre se genere el bloque div con los botones, ocultándolo en el caso de que el mensaje esté ya puntuado. De esta manera, a través de JavaScript, cuando se pulse el botón para modificar nota, solamente habrá que *desocultarlo*.

El siguiente paso ha sido decidir qué elemento poner como botón, si una imagen, un enlace, un form, un párrafo… Finalmente se ha utilizado un enlace (<a>), con el script como valor del campo *href*.

Los iconos que se muestran en StatusNet, se obtienen de una imagen única, con todos los iconos juntos, y seleccionando el que se desee utilizando la propiedad b*ackground-position*.
Para no utilizar esta imagen, que ocupa demasiado, y para aislar todo lo necesario de Grades en su propia carpeta, se ha diseñado el icono aparte, y se le ha colocado al enlace con la propiedad *background* de CSS:

El siguiente paso y más importante, es lograr la funcionalidad que se desea al pulsar ese enlace. Para ello, se ha creado una función JavaScript "editarNota", y se le ha referenciado desde el campo *href*:

```
$item->out->elementStart('a', array('href' => 'javascript:editarNota(' . $noticeid . ');', 'class' => 'notice-modify-grade', 'id' => 'button-modify-grade-' . $noticeid));
```

En esta función JavaScript no basta con hacer visible una clase determinada, ya que de esa manera, todos los bloques div de la página se mostrarían. Por ello, ha sido necesario modificar de nuevo la función *onEndShowNoticeItem*, añadiendo un ID personalizado con el *noticeid* al div que alberga los botones.

De esta manera, pasando el ID de la noticia a la función JavaScript, se podrá realizar acciones sobre un elemento determinado, sin afectar al resto de la página.

El paso de parámetros desde PHP a JavaScript se ha realizado de la siguiente manera:

```
'href' => 'javascript:editarNota(' . $noticeid . ');'
```

A continuación, en *GradesPlugin.php* es necesario utilizar el *hookonEndShowScripts* para cargar el fichero *grades.js* con el script JavaScript que se ha desarrollado:

```
function onEndShowScripts($action) {
	$action->script($this->path('js/grades.js'));
	return true;
}
```

Sin embargo, al aplicarse el script, los botones se muestran más a la izquierda de lo normal, provocado por el nuevo botón de modificar nota.

Por tanto, se le ha asignado un ID también al enlace de modificar nota, de manera que cuando se le pulsa, además de mostrar los botones, el propio enlace se oculta.

El código completo de la función JavaScript desarrollada es el siguiente:

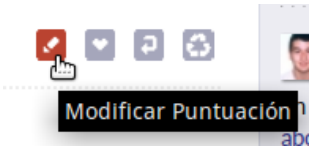
```
function editarNota(id) {

   $("#div-grades-hidden-" + id).attr('class', 'notice-grades');
   $("#button-modify-grade-" + id).attr('class', 'notice-modify-grade-hidden');
}
```

Otro detalle, es que se ha modificado el color del botón para modificar nota, ya que era difícil localizarlo. Todas las imágenes que se han desarrollado utilizando el software Adobe PhotoShop, pierden los permisos al pasarlas al sistema Linux, por lo que ha sido necesario aplicarles una serie de permisos que permitieran su lectura, ya que de lo contrario no se mostraban en la página.

```
chmod a+r nameFile
chmod g+w nameFile
```

Se ha añadido también un *title* al botón, de manera que si se sitúa el ratón encima, se obtenga información de qué realiza ese enlace.

El resultado final es el siguiente:



Añadir información de Profesor en sus tweets

Ficheros modificados: GradesPlugin.php, grades.css

Para poder distinguir más fácilmente los tweets de profesores entre el resto de los mensajes del sistema, es necesario que sus anotaciones tengan algo diferente.

Se han barajado distintas opciones:

Ninguna de ellas es especialmente llamativa, la primera, porque recarga demasiado el *timeline*, y la segunda, porque en cuanto se modifique el avatar por defecto por una imagen no se va a distinguir.

Por ello, se ha optado por una opción bastante más compleja y llamativa. La idea es añadir una imagen de un birrete a los avatares de los mensajes realizados por profesores. Para ello, se ha realizado el diseño con Adobe PhotoShop, y se ha incluido un nuevo elemento imagen en todos los *tweets* realizados por usuarios con el rango *grader*.

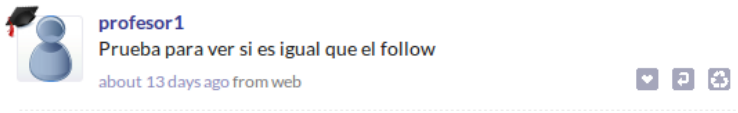
Se ha modificado por tanto la función *onStartShowNoticeItem* de *GradesPlugin.php* para añadir esta comprobación:

```
// Si la noticia es de un profesor, mostramos el birrete.
if ($args->notice->getProfile()->getUser()->hasRole('grader')) {

$path = $this->path('css/birrete-small.png');
$args->out->element('img', array('id' => 'birrete-grades', 'alt' => 'Profesor', 'src' => $path));
```

Una vez añadida la imagen, a través de reglas CSS se ha modificado el aspecto para colocarla encima del avatar, y con una cierta inclinación.

```
float: left;
z-index: 100;
position: absolute;
left: -9px;
rotation: -18deg;
moz-transform: rotate(-18deg);
webkit-transform: rotate(-18deg);
```

El resultado es el siguiente:



Sin embargo, en los mensajes sin avatar, por ejemplo al acceder al perfil de un profesor, se muestra un birrete que en este caso no debería.

Para solucionarlo, a través de reglas CSS y conociendo el ID del body cuando se accede a la acción "perfil" (*showstream)*, se ha conseguido eliminar.

```
.notice-source-activity #birrete-grades, #showstream #birrete-grades
{
	display:none;
}
```

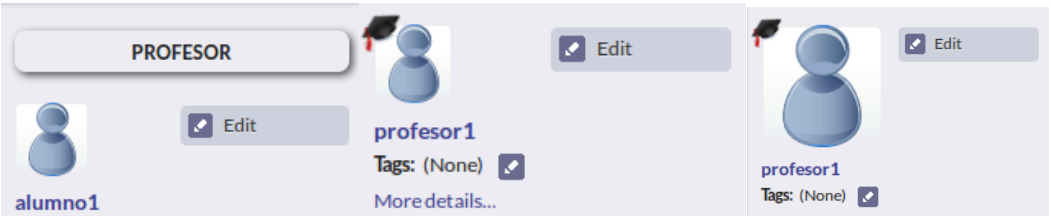
Además, los mensajes de tipo *activity*, como el que se muestra a continuación en el que se incluye información sobre eventos del sistema, nunca va a ser necesario que se muestre el birrete, por lo que como tienen una clase especial, *.notice-source-activity*, también se ha conseguido eliminar.

Añadir información en el perfil de los profesores

Ficheros modificados: GradesPlugin.php, grades.css

Al entrar en el perfil de un usuario con rango *grader*, es necesario conocer que ese usuario es un profesor.

Por ello, tras encontrar el *hook* que permitiera insertar un elemento en el *aside* del perfil, *onStartShowAccountProfileBlock*, se han barajado distintas opciones:



A pesar de ser más atractiva visualmente la del birrete, el problema es que cuando se accede al perfil del usuario el tamaño del avatar varía, y por tanto el birrete queda demasiado pequeño.

Además, el *hook* utilizado en ambos casos es el mismo, por lo que no es posible poner un birrete de distinto tamaño dependiendo del *hook*.

Finalmente, y para evitar esto, se ha optado por realizar una mezcla de ambas opciones, de manera que ahora el tamaño del avatar no influye en nada.

Todas las modificaciones se han realizado en la función *onStartShowAccountProfileBlock* del archivo *GradesPlugin.php*, y en el fichero *grades.css*.

Los botones para puntuar se ven mal en el móvil

Ficheros modificados: mp-screen.css

La parte del *StatusNet* que se encarga de modificar la apariencia visual para la versión móvil, es el plugin *MobileProfile*.

Como es este plugin el que se ejecuta al detectar un dispositivo móvil, y son sus CSS las que se aplican, es necesario modificarlas y añadir nuevas reglas para adaptar Grades a dispositivos móviles.

En este caso, no es posible realizar esto desde el propio plugin Grades.

Se han añadido reglas para eliminar el margen izquierdo de los botones y para disminuir el derecho, de manera que todos quepan en la pantalla.

Por otro lado, al botón 0, situado a la derecha del bloque de botones, se le ha aplicado un margen derecho para poder centrarlo.

```
.notice .notice-grades .form_grade input.submit{
    margin-left: 0px;
    margin-right: 10px;
}

.notice .notice-grades #grade-0 input.submit{
    margin-right:40px;
}
```

Los Graders deben tener un contexto asociado

Ficheros modificados: Gradesgroup.php, GradesPlugin.php

Una de las necesidades y de los cambios más importantes en Grades, es la existencia del contexto relativo a profesores.

En la versión anterior, cualquier profesor podría puntuar cualquier tweet, independientemente de quien fuera. Este comportamiento no es el deseado, ya que un profesor únicamente debería poder puntuar los mensajes creados por sus alumnos, es decir, los mensajes destinados a los grupos de los cuales ese usuario es profesor.

Con la estructura actual de Grades, y con las tablas utilizadas, esto no es posible, pues de alguna manera se deben vincular de manera persistente los profesores con sus grupos.

Con la nueva estructura, la idea es que un profesor pertenezca a 0…n grupos, y pueda puntuar únicamente mensajes dirigidos a esos grupos.

Para conseguir esto, es necesario generar una nueva clase que represente a la nueva tabla de la base de datos, y realizar todas las funciones necesarias para lanzar consultas sobre la misma.

Se ha creado el archivo *Gradesgroup.php*, que representará la nueva tabla.

Tras estudiar el formato de la misma, se ha diseñado de manera que tenga las siguientes columnas:

- ID del usuario
- ID del grupo asociado

En cuanto a la clave primaria, como un profesor puede tener más de un grupo, no puede ser la ID del profesor, ya que se limitaría a 1 los grupos asociados a los profesores, por lo que se ha escogido como clave primaria el conjunto de ambas.

Por otro lado, se han creado los métodos de clase staticGet, pkeyGet, y la función schemaDef, que define el esquema de la tabla (y que creará la tabla en la base de datos la primera vez que se ejecute, por lo que debe representar la estructura correctamente).

Hasta ahora, para que un usuario se convirtiera en profesor, había que asignarle a través del script *userrole.php* el rango "grader". Sin embargo, ahora además de eso va a ser necesario vincularles a los grupos.

Por ello, se ha desarrollado la función *vincularGrupo*, que recibe el userID y el groupID y los añade a la tabla. También se ha creado la función *desvincularGrupo* que realiza justamente lo contrario.

Al añadir una nueva clase que representa una nueva tabla, es necesario indicárselo al plugin para que comprueba su estructura, por lo que se ha modificado la función oncheckSchema para añadir lo siguiente:

```
$schema->ensureTable('grades_group', Gradesgroup::schemaDef());
```

Crear script para crear Grader

Ficheros modificados: createGrader.php

Al añadir contexto a los profesores, es necesario además de añadirle el rango grader, vinculares al grupo que se desee. Por ello, se ha tenido que desarrollar un script PHP para que a través de parámetros se pueda crear un profesor completo, es decir, asignarle el rango grader, y vincularle a un grupo.

Para ello, se ha hecho uso de las funciones anteriormente creadas, vincularGrupo y desvincularGrupo.

Además, otro detalle interesante a mencionar, es que como en la nueva versión de BoloTweet los usuarios por defecto no pueden eliminar sus mensajes, es también necesario en este script añadir el rango *deleter* a los profesores, para permitir que ellos sí puedan borrar sus propios mensajes.

En el script también se ofrece una opción "delete", que realiza justamente lo contrario. Se le quita el rango *grader* al usuario, se le desvincula del grupo que se haya pasado como parámetro, y se le elimina el rango *deleter*.

El script creado se llama *createGrader.php*, y se puede encontrar en la carpeta scripts de BoloTweet.

Al existir puntuaciones múltiples, el resultado debe ser una combinación de todas

Ficheros modificados: Grades.php, GradesPlugin.php

En la nueva versión de BoloTweet, es posible que más de un profesor puntúe un mismo tweet. Esto provoca que el mecanismo de calificaciones varíe para soportar puntuaciones múltiples.

Entre otras cosas, para poder escoger una puntuación total para una noticia concreta, es necesario realizar algún tipo de tratamiento para las puntuaciones asociadas a un mismo tweet.

Para soportar esto, lo primero que se ha hecho es crear la función getNoticeGradesAndGraders.

Esta función lo que hace es buscar todas las notas y graders para un noticeID dado. Si encuentra más de una, las almacena en un array asociativo. Si no encuentra ninguna, devuelve '?'.

Como en el caso de que exista más de una nota para un mismo tweet, se debe hacer un tratamiento especial, se ha creado una función denominada *devolverGrade*, que tras pasarle un *noticeID*, devuelve directamente el contenido que se debe mostrar en el recuadro de la puntuación.

El funcionamiento de esta nueva función es el siguiente:

- *devolverGrade* llama a la función getNoticeGradesAndGraders, y procesa su resultado.
- Si el resultado es '?', quiere decir que no hay ninguna nota, y por tanto se devuelve '?'.
- Si el array es de 1 elemento, quiere decir que solamente hay una nota, y por tanto se devuelve un array asociativo con el nombre del profesor, y la nota asociada.
- Por último, si tiene más de 1 elemento, se deben tratar las puntuaciones, devolviendo un array asociativo con "Nota", como nombre del profesor, y el resultado del tratamiento como puntuación. Por ello, la función *devolverGrade* acepta un segundo parámetro que va a ser el tipo de procesamiento que se quiere (media, mediana…).

Por otro lado, en la función *onStartShowNoticeItem*, tras llamar a devolverGrade, únicamente se debe comprobar si el resultado es un array (de lo contrario es un '?', y no hay que mostrar nada), y si lo es, se muestra tal cual, ya que el resultado obtenido está preparado para ser mostrado sin tratamiento alguno.

Un detalle interesante, es que al enviar el resultado en un array asociativo, la manera de acceder al contenido del array es más compleja de lo habitual.

```
$gradeValue = reset($gradeResult);
$grader = key($gradeResult);
```

El comando *reset* pone el puntero al comienzo del array, y devuelve el contenido de la primera posición. Por otro lado, *key*, devuelve la clave del puntero actual, pudiendo obtener así ambos valores.

Ocultar botones en mensajes puntuados

Ficheros modificados: GradesPlugin.php, Grades.php

Antes de existir puntuaciones múltiples, los mensajes puntuados ya no mostraban los botones. Sin embargo, al soportar puntuaciones múltiples, ahora sí puede ser necesario mostrar el bloque con los botones aunque el tweet esté puntuado, ya que puede haber otro profesor que aún no haya calificado ese mensaje.

Para lograr este comportamiento, en la función *onEndShowNoticeItem* se han realizado los siguientes pasos:

- En primer lugar, se comprueba que el tweet no sea de un grader, ya que estos no se pueden puntuar.
- A continuación se comprueba si el usuario actual es un grader válido para ese tweet. Esto significa, que sea grader, y que esté vinculado con el grupo en el que está publicado el tweet. Para esto, se ha creado la función *getValidGrader*.
- El siguiente paso es obtener la nota actual del tweet, utilizando la función *getNoticeGrade*, que se ha tenido que modificar para incluir como parámetro el *nickname*, y comprobar así si el usuario actual ha puntuado el tweet.
    - Si la nota es '?', quiere decir que el profesor aún no ha puntuado ese tweet, y por tanto se crea el bloque con los botones.
    - Si por el contrario la nota es distinta de '?', quiere decir que ese profesor ya ha puntuado ese mensaje, y por tanto se crea el bloque con los botones oculto.

El mecanismo de ocultar y mostrar este bloque con botones se ha explicado con detalle en una tarea anterior, *Botón para modificar las puntuaciones*.

Grade Reports no funciona con puntuaciones múltiples

Ficheros modificados: GradereportAction.php, Grades.php

Al ofrecer la posibilidad de que un tweet pueda ser puntuado por más de un profesor, y tal y como está implementada la consulta que obtiene el listado de alumnos y puntuaciones para un grupo, los resultados obtenidos son erróneos.

Esto es, porque actualmente se realiza un *sum* de las puntuaciones para un determinado usuario, de manera que actualmente si un tweet ha sido puntuado por dos profesores, con un 3 y con un 2, esta consulta devolverá un 5, cuando la puntuación real del tweet es un 2,5.

Si se quiere poder realizar cualquier tipo de tratamiento con las puntuaciones de un mismo tweet (media, mediana…), entonces la solución necesitaría que se devolviera el listado completo de la tabla grades, y posteriormente realizar el tratamiento desde PHP, haciendo los cálculos y agrupando por usuarios.

Este proceso sería demasiado costoso, por lo que se ha decidido implementar con una consulta bastante compleja, que devuelva directamente el listado correcto de usuario más puntuación, pero teniendo en cuenta que únicamente funcionará si como tratamiento de puntuaciones múltiples se ha escogido media.

Esto es, por que SQL no dispone de todas las operaciones matemáticas, por lo que es probable que el tratamiento que se haya escogido no se pueda realizar a través de un comando SQL.

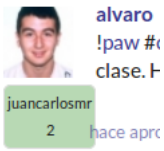
La consulta que se ha implementado realiza los siguientes pasos:

- En primer lugar se hace una selección de los tweets de un usuario dado un grupo determinado. Esa recopilación de tweets se agrupan por ID, de manera que una vez asociados, se puede aplicar el tratamiento escogido para puntuaciones múltiples, en este caso a través de la función *mean* de *SQL*.
- Con esta consulta, se habrían obtenido las notas medías por tweets. Como lo que se necesita son las puntuaciones por usuario, se ha realizado otra consulta por encima de la anterior, de manera que utilizando el resultado de esta, agrupe por *nickname*, es decir, por usuario, y sume todas las puntuaciones acumuladas de sus tweets.
- Con esto se conseguirá devolver un listado de usuarios relativos a un grupo, junto con su puntuación total real.

```
select tmp.nickname as userid, sum(tmp.gradeAvg) as total
from  (select p.nickname, avg(g.grade) as gradeAvg
from grades g, group_inbox gr, notice n, profile p
where g.noticeid = gr.notice_id
and gr.group_id = idDelGrupo
and g.noticeid = n.id
and n.profile_id = p.id
group by g.noticeid) as tmp
group by tmp.nickname
```

Nombres largos de profesor descuadran recuadro con puntuación

Ficheros modificados: grades.css

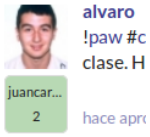


Para arreglar esto, lo que se ha hecho es añadir una serie de reglas CSS que lo controlen.

- Se ha definido un tamaño fijo, para evitar que el recuadro crezca más de lo esperado.
- Lo siguiente que se ha hecho es colocar el parámetro *overflow* como *hidden*, y que en el caso de que se produzca ese solapamiento, se coloquen automáticamente puntos suspensivos.

```
white-space: nowrap;
overflow: hidden;
text-overflow: ellipsis;
width: 44px;
```

El resultado ha sido el siguiente:



Actualizar mecanismo para actualizar nota

Ficheros modificados: GradeAction.php, Grades.php

Al existir puntuaciones múltiples, la función para actualizar la nota, que anteriormente recibía el *noticeid*, y la nueva nota, ya no funciona, pues ahora es necesario indicar también el profesor asociado.

En GradeAction, la comprobación de si un tweet estaba o no, puntuado, se ha modificado para utilizar la nueva función getNoticeGrade, que ahora también recibe el ID de usuario.

La función updateNotice de Grades se ha tenido que modificar, añadiendo un nuevo parámetro que sea el *userid*, y modificando la consulta para actualizar únicamente el registro en el que coincidan el noticeid y el userid con los pasados a través de parámetros.

```
$qry = 'UPDATE ' . $user_table .
                ' SET grade=' . $grade .
                ', cdate=\'' . $time . '\'' .
                ' WHERE noticeid=' . $noticeid .
                ' AND userid="' . $userid . '"';
```

Mostrar en el perfil de los grupos un apartado con los profesores

Ficheros modificados: GradesPlugin.php, groupaction.php, Gradesgroup.php

Al visualizar los perfiles de los grupos, es conveniente indicar en algún apartado quiénes son los profesores del mismo.

Para ello, lo primero que se ha hecho es buscar el *hook* que permitiera incluir un nuevo apartado con profesores en el perfil de los grupos, obteniendo como resultado que no hay ninguno que lo permita.

Para solucionar esto, y ya que no tendría sentido *hardcodear* código de Grades en el *core* de StatusNet, lo que se ha hecho es crear manualmente el hook, para posteriormente poder utilizarlo desde Grades.

En el archivo */lib/groupaction.php*, se ha localizado la zona donde se requería añadir un *hook*, en este caso justo antes y después de insertar secciones en el *aside* del perfil de los grupos, y se ha añadido el siguiente código:

```
Event::handle('StartAsideGroupProfile', array($this,$this->group));

Event::handle('EndAsideGroupProfile', array($this, $this->group));
```

De esta manera, se podrán utilizar estos dos nuevos *hooks* para insertar elementos al comienzo de las secciones, y al final de las mismas. En el *hook* es necesario pasarle como parámetro $this, ya que posteriormente será utilizado en GradesPlugin para añadir elementos, y $this->group para obtener el grupo en el cual buscar los profesores.

Una vez creados los *enganches*, se ha creado la correspondiente función en GradesPlugin utilizándolos:

```
function onStartAsideGroupProfile($out) { … }
```

Para crear una sección similar a las ya existentes en el perfil de los grupos, como "Miembros" o "Admins", se ha estudiado su código para recrear de la mejor forma posible la nueva sección.

Por otro lado, para obtener los profesores de un grupo, se ha creado la función getGraders.
Es a esta función a la que se llama junto con el ID del grupo en cuestión, para obtener los perfiles de los profesores vinculados al grupo, necesarios para generar posteriormente la lista de avatares.

Por último, se ha generado el título de la sección junto con el número total de profesores, y se ha generado la lista de profesores del siguiente modo:

```
        $out->elementStart('div', array('id' => 'div-members-scroll'));
              $gmml = new GroupMembersMiniList($graders, $out);
              $cnt = $gmml->show();
              if ($cnt == 0) {
                  $out->element('p', null, _('(Ninguno)'));
}
```

El resultado final es el siguiente:

Al puntuar un tweet se redirige a la página del perfil actual

Ficheros modificados: GradeAction.php, GradeForm.php, GradesPlugin.php, grades.js, grades.css

Para entender este problema, es preciso explicar rápidamente el funcionamiento básico de los formularios y sus acciones. Cuando se genera un formulario, en este caso cada botón lo es, se le indica una acción asociada. Esta acción se corresponde con una nueva página web, con una nueva URL asociada. Es por esto, por lo que los cambios que se realicen desde esta última acción, se realizan sobre la nueva página, no sobre desde la que se accede al formulario.

El problema que sucede es que al pulsar un botón para puntuar, realmente se está accediendo a otra nueva página, en este caso sin contenido, ya que la acción asociada a los botones simplemente realizar cambios internos, por lo que o bien se redirige a otra página, o bien se muestra una página en blanco.

Para solucionar esto sin utilizar ninguna de estas opciones, han sido necesarios varias semanas de prueba y error, y de estudio acerca del comportamiento de otros plugins en el sistema.

Finalmente, se pudo entender el funcionamiento de un mecanismo utilizado en otros plugins, que lo que hace es permitir ejecutar la nueva acción sin moverse de la página actual, y que al final de esta tarea de explicará cómo se ha utilizado.

El esquema que permite esto es el siguiente:

```
if ($this->boolean('ajax')) {

    $this->startHTML('application/xml,text/xml;charset=utf-8');
    $this->elementStart('head');
    $this->element('title', null, _('Disfavor favorite'));
    $this->elementEnd('head');
    $this->elementStart('body');
    $this->element('p',  array('class'  =>  'notice-current-grade-value'),  '  '  .
    $gradevalue);
    $this->elementEnd('body');
    $this->elementEnd('html');
}
```

Al aplicar este esquema en la acción de destino, se consigue recargar únicamente el formulario desde el que se ha accedido, dejando el resto de la página intacta.

Sin embargo, en este caso, cuando se pulsa por ejemplo el botón 2, lo único que se recargaría sería ese botón, siendo imposible modificar el resto de la página, y siendo por tanto imposible quitar el bloque completo de botones, ni siquiera con JavaScript, ya que cualquier script que se ejecute junto con ese esquema Ajax en la acción destino, resulta en un error.

Tras muchísimas pruebas, en un primer momento se consiguió una solución temporal, pero aceptable.

Lo que se hizo fue almacenar la URL actual antes de llamar a la nueva acción vinculada con el formulario, de manera que sea a esta URL a la que se redirija al terminar la acción.

Para almacenar la URL actual en PHP, se ha realizado del siguiente modo, utilizando variables de servidor:

```
$url = "http://".$_SERVER["SERVER_NAME"].$_SERVER["REQUEST_URI"];
```

El siguiente paso, ha sido modificar el formulario actual para poder pasar a través de él esta URL almacenada. Por ello, se ha modificado el archivo GradeForm, añadiéndole un nuevo atributo url. Para poder pasar la URL a la acción, también es necesario añadir un nuevo elemento *hidden*, en el que se pase como valor la URL.

Por último, se ha hecho que la redirección que antes se hacía a una página fija, ahora se haga a la página en la que se estaba puntuando.

```
$url = $this->trimmed('url');
common_redirect ($url,303);
```

Este mecanismo funciona, pero tiene algunos inconvenientes. El más importante es que al recargarse la página, se cambia la posición al inicio de la misma.

Para mejorar este comportamiento, y tras muchas búsquedas, se ha utilizado un ID en la URL, para poder recargar la página y moverla a la posición del ID.
En este caso, si se coloca el ID de la noticia que se ha puntuado, se conseguirá mover la página hasta ese punto.

```
$url = $this->trimmed('url');
$url .= '#notice-' . $noticeid;
common_redirect($url, 303);
```

En esta modificación existe un problema principal, y es que el protocolo de la URL está escrito manualmente (http://), por lo que al variar entre desarrollo local y desarrollo en servidor, es necesario modificarlo. Para evitar esto, se ha generado la URL con la función *common_path*, de manera que siempre se obtenga el protocolo correcto.

El problema es que tanto common_path como _SERVER["REQUEST_URI"] devuelven el nombre del servidor en la respuesta, generando esa parte duplicada, y por tanto una URL inexistente. Por ello, con la función *substr* de PHP, se ha tenido que eliminar de una de las dos:

```
$request = substr($_SERVER["REQUEST_URI"],11);
$url = common_path().$request;
```

Hasta aquí sería la primera versión que se ha utilizado para intentar corregir esta redirección al puntuar.

Tras barajar otras posibilidades, y para evitar el paso de la URL a través del formulario, se optó por utilizar otra variable del servidor, HTTP_REFERER, que almacena la URL del recurso anterior a una llamada, en este caso, la página en la que se está puntuando.

Se modificó de nuevo todo el mecanismo de redirección, utilizando en este caso la URL proporcionada por dicha variable.

```
$url = $_SERVER[“HTTP_REFERER”];
$url .= '#notice-' . $noticeid;
common_redirect($url, 303);
```

A pesar de ser una solución aceptable, para los profesores era incómodo el refresco de la página por cada puntuación, además de resultar confuso el movimiento de la página hasta la noticia puntuada.

Por ello, se ha producido una nueva modificación para intentar evitar el *refresh* de la página, en este caso siendo la definitiva.

La única manera de mantenerse en la página desde la que se realiza una acción, es utilizando el mecanismo AJAX comentado anteriormente. Por ello, en GradeAction, se ha eliminado el mecanismo de redirecciones, y se ha vuelto a añadir el esquema AJAX.

Tras diversas pruebas, la idea es utilizar este mecanismo para mantenerse en la página, y posteriormente con ayuda de JavaScript realizar algunas acciones sobre la misma.

Como desde la acción no es posible aplicar ningún script, lo que se ha hecho es realizarlo antes de llamarla, es decir, en el momento que se pulsa el botón con la puntuación.

Como del esquema Ajax lo único que nos interesa es que se mantenga en la página actual, en el esquema Ajax se ha indicado que el formulario se sustituya por un elemento párrafo (<p>) vacío, ya que el comportamiento se va a realizar desde JavaScript.

Para asociar un script al formulario, se ha utilizado el evento *onclick*, ya que como se ejecuta directamente en el lado del cliente, va a ser posible realizar algunos cambios en la página, y de fondo realizar la acción asociada al formulario.

Por ello, se ha modificado el GradeForm, y ya que la función predefinida submit no acepta más parámetros, se ha tenido que eliminar esta función, y generar un elemento submit manualmente.

```
$this->out->element(‘input’, array(‘type’ => ‘submit’,
‘id’ => ‘grade-submit-‘ . $this->notice->id,
‘class’ => ‘submit’,
‘value’ => ‘ . $this->value,
‘title’ => ‘Grade this notice’,
‘onClick’ => ‘puntuarNota(‘.$this->notice->id.’,’.$this->value.’);’));
```

Como se puede ver, se le ha asociado un evento onClick, en el que se llama a la función puntuarNota, y a la que se le pasa el *noticeid* y la puntuación.

Para que se entre en el esquema Ajax, es necesario modificar el formulario para añadirle la clase Ajax.

```
function formClass() {

return ‘form_grade ajax’;

    }
```

El último paso ha sido desarrollar el script. Como el script recibe el *noticeid* y la puntuación, se podrá seleccionar desde JavaScript la noticia puntuada, y con el valor de la nota mostrar algún tipo de mensaje.

Como la idea es eliminar los botones una vez puntuado el mensaje, a través de JQuery, con la ayuda de selectores CSS, y una vez obtenida la noticia puntuada, se ha podido acceder al bloque "notice-grades", que es el que alberga todos los botones.

Una vez localizado ese div, se debe ocultar desde JavaScript. Es importante que se oculte con CSS, y no que se elimine el elemento, ya que de lo contrario, al ejecutarse antes de llamar a la acción, ésta nunca se llegará a ejecutar.

Para ocultar el bloque desde JavaScript lo que se ha hecho es asignarle una clase determinada, la cual tiene asociada diversas reglas CSS para que no se muestre.

```
$("#notice-"+noticeid+'>.notice-grades').attr('class','notice-grades-hidden')
```

Una vez ocultados los botones, lo que se ha hecho es mostrar un mensaje indicando que ese tweet ha sido puntuado, de manera que se pueda identificar rápidamente en una página los tweets ya calificados.

Se podría haber actualizado directamente el recuadro con la nota con la nueva puntuación, sin embargo, no ha sido posible realizarlo por las puntuaciones múltiples.

Al soportarse más de una puntuación para un mismo tweet, la puntuación final no se puede calcular correctamente, ya que aunque se hiciera la media con la puntuación que hubiera anteriormente, el resultado no sería el correcto.

Por ejemplo, si un tweet ha sido puntuado por tres profesores con las siguientes notas, la calificación final sería de 2.

$$(3 + 2 +1) / 3 = 2$$

Si el mismo tweet ha sido calificado por 3 profesores, pero al puntuar el último profesor con un 1, tan sólo se hace la media con la nota media anterior, 2.5, el resultado final no es el correcto.

$$(3 + 2) / 2 = (2.5 + 1) / 2 = 1.75$$

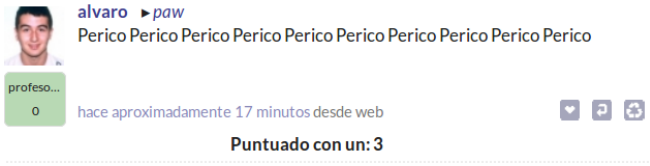
Por ello, se ha optado por utilizar la otra opción, y generar un párrafo indicándolo:

```
$("#notice-"+noticeid).append('<p class="temp-text-grades">Puntuado con un: '
+ value + '</p>');
```

Como se puede observar, se le ha asignado una clase determinada para poder aplicarle un cierto estilo (texto centrado, en negrita, y situado al final del tweet).

```
.temp-text-grades{

    margin-left: 35%;
    float: left;
    font-weight: bold;

}
```

Con todos estos pasos, lo que se ha conseguido es que cuando un profesor puntúe un tweet, automáticamente se ocultarán los botones, se mostrará un mensaje indicando la calificación, y lo más importante, no se recargará la página, pudiendo puntuar con una gran comodidad.



Desglose de notas en tweets con múltiples puntuaciones

Ficheros modificados: GradesPlugin.php, grades.js, grades.css

Al existir puntuaciones múltiples, no es posible mostrar en el recuadro con las puntuaciones el listado completo de profesores junto con sus puntuaciones, ya que por espacio, esto no es posible.

Sin embargo, puede ser interesante que los alumnos sepan qué profesor le ha puntuado qué nota, por lo que se ha añadido un método de desglose en el que se pueda ver la lista completa.

Para conseguir esto, se han realizado los siguientes pasos:

- En primer lugar, para saber si hace falta o no el desglose, hay que comprobar si el tweet contiene puntuaciones múltiples.

  Para ello, es tan sencillo como comparar el nombre del *grader*. Si el nombre del *grader* es igual a "Nota", quiere decir que hay más de una puntuación.

- El siguiente paso ha sido obtener la lista de puntuaciones y de profesores asociadas a ese tweet utilizando la función getNoticeGradesAndGraders.

- Una vez obtenido el array asociativo, es necesario recorrerlo e ir incluyendo los valores en un bloque div, que posteriormente representará al desglose de puntuaciones.

```
foreach ($gradesAndGraders as $profesor => $nota)
```

- Para mostrar ese desglose, en los mensajes con puntuaciones múltiples se ha añadido un evento *onclick* al recuadro con la puntuación.

```
$args->out->elementStart('div', array('class' => 'notice-current-grade',
'onclick' => 'mostrarPuntuacion(' . $noticeid . ');'));
```

- Lo siguiente que se ha hecho es crear una serie de reglas CSS que le proporcionen una apariencia correcta a este bloque.

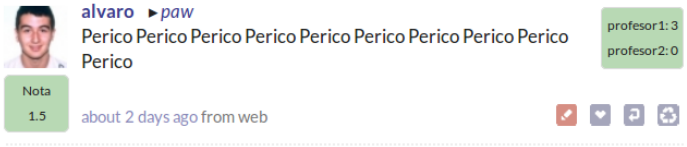
```
position: relative;
float: right;
background-color: rgb(184, 217, 180);
border-radius: 4px;
padding: 4px;
top: 8px;
border: 1px solid;
border-color: rgb(220, 220, 220);
font-size: 10px;
```

- Por último, se ha desarrollado el script JavaScript, que se encarga de mostrar u ocultar el bloque dependiendo de si está, o no visible.

  Para ello, con la propiedad *length* de *JQuery*, se ha comprobado si existe un elemento con la clase *.div-with-grades-hidden*, y de ser así, se cambia la clase a *.div-with-grades*, para que se le apliquen las reglas mostradas anteriormente.

  Si por el contrario existe una clase del tipo *.div-with-grades*, se le asigna la clase oculta, *.div-with-grades-hidden*.

El resultado final del desglose es el siguiente:



Optimizado comportamiento de la función devolverGrade

Ficheros modificados: Grades.php, GradesPlugin.php

En la función *onStartShowNoticeItem*, encargada de mostrar el recuadro con las notas al cargarse los mensajes, anteriormente se llamaba a *devolverGrade* para obtener la nota asociada al tweet, y si el mensaje tenía más de una puntuación, se llamaba a la función *getNoticeGradesAndGraders* para obtener el listado de puntuaciones completo (profesor más nota).

Sin embargo, la función *devolverGrade* ya llama a *getNoticeGradesAndGraders*, por lo que no tiene sentido realizar esta llamada dos veces, ya que esto se traduce en más tiempo y más conexiones a la base de datos.

Por tanto, se ha modificado la estructura, de manera que ahora se llama a getNoticeGradesAndGraders desde onStartShowNoticeItem una sola vez, haga falta o no desglose.

Se ha cambiado por tanto el funcionamiento de *devolverGrade*, ya que ahora recibe el resultado de *getNoticeGradesAndGraders* a través de un parámetro, y por tanto se ha eliminado la llamada a esa función.

De esta manera, en onStartShowNoticeItem, primero se llama a getNoticeGradesAndGraders, y su resultado se le pasa a devolverGrade, para que haga el procesado de las puntuaciones, y devuelva la cadena que se debe mostrar en el recuadro con la puntuación.

La diferencia es que si el nombre del *grader* es igual a "Nota", como ya se ha llamado en onStartShowNoticeItem a getNoticeGradesAndGraders, no hace falta llamarla de nuevo, ya que se dispone del array asociativo con las puntuaciones, evitando así una llamada innecesaria a la base de datos.

Cambiar formato del listado GradeReports.

Ficheros modificados: Grades.php, grades.css, grades.js, GradereportAction.php

La implementación del Grade Reports de la versión anterior, era una página en la que se mostraban todos los grupos existentes en BoloTweet, y cada uno de ellos con un listado (sin ordenar) de sus alumnos junto con sus puntuaciones, y con una extensión tan larga como alumnos y grupos existieran en BoloTweet.

La idea es modificar este comportamiento en varios aspectos principales:

- En el Grade Reports únicamente se podrán ver los grupos a los que pertenezca el usuario actual. No tiene sentido poder ver las notas del resto de grupos de BoloTweet.
- El listado de alumnos de cada grupo estará ordenado de mayor a menor puntuación total, y en él se mostrarán todos los usuarios del grupo, tengan o no puntuaciones.
- Los alumnos podrán visualizar el listado de igual manera que los profesores, con la única diferencia que los alumnos no verán las puntuaciones asociadas a cada usuario.

La apariencia anterior era la siguiente:


Grade Reports

Desarrollo de Aplicaciones y Sistemas Inteligentes

- dlora, 2.5000
- javiem04, 2.0000
- jdapaz, 3.0000
- joelsanc, 2.0000
- masotelo, 3.0000
- pablocerrocanizares, 3.0000

Sistemas web

- aleramon, 2.5000
- ccgarcia, 2.0000
- cghernandezgil, 2.0000
- cmperdomo, 2.2000
- croa, 2.7500
- glcita, 0.7500
- hectorjesusfigueroa, 2.5000
- ignaciobartolome, 0.6667
- ivmendez, 3.0000
- jmrnavarro, 2.0000
- jolmedo, 2.7500
- jorgecollado, 1.5000
- luisfema, 2.1667
- luislaza, 2.0000

Modificaciones GradeReports

Se ha modificado la forma de obtener los grupos, ahora en vez de coger todos los existentes en BoloTweet, se han seleccionado únicamente a los que pertenece el usuario.

```
$groupsUser = $this->user->getGroups()->fetchAll();
```

Posteriormente, se ha utilizado la función *getGradedNoticesAndUsersWithinGroup*, que tal y como se explicó en la tarea *Grade Reports no funciona con puntuaciones múltiples*, se ha modificado para soportar las puntuaciones múltiples, y se ha obtenido por tanto el listado de usuarios y puntuaciones totales relativos a cada grupo.

El siguiente paso ha sido modificar la visualización de todos los grupos, y del listado en sí, ya que a pesar de que actualmente sólo se muestren los grupos a los que pertenece el usuario, es poco agradable visualmente el disponer de una lista tan extensa.

- Los nombres de los grupos ahora son encabezados del tipo 3, <h3>, y subrayados.
- Cada listado de cada grupo se ha incluido en un div oculto, que únicamente se mostrará si el usuario desea expandirlo.
- El enlace "Expandir" se ha convertido en un botón únicamente a través de reglas CSS.

El siguiente paso ha sido desarrollar el JavaScript asociado a los botones expandir. Un detalle importante es que como cada botón únicamente debe expandir su grupo, ha sido necesario añadir identificadores a los bloques DIV, para poder seleccionar elementos concretos desde el script.

De esta manera, se ha implementado la función JavaScript *mostrarReport*, la cual recibe el ID del grupo, y cuyo contenido es el siguiente:

```
if($("#grade-report-group-" + groupid + '>.grade-show-report').text() == 'Expandir'){

$("#grade-report-group-" + groupid + '>.report-group-hidden').toggle('fade',300);
$("#grade-report-group-" + groupid + '>.grade-show-report').text('Ocultar');
        }

else {
$("#grade-report-group-" + groupid + '>.report-group-hidden').toggle('fade',300);
$("#grade-report-group-" + groupid + '>.grade-show-report').text('Expandir');

        }
```

Para ocultar por defecto los grupos, se les ha asignado la clase *.report-group-hidden*, que tiene asociadas unas reglas CSS para ocultarlos.

De esta manera, al entrar en GradeReports, se verá el encabezado de los grupos, pero ningún listado. Para expandir cualquiera de ellos, se ha utilizado la función de JQuery *toogle*, junto con el parámetro *fade*.

Esto lo que hace es que además de añadir un desvanecido, se encarga automáticamente de mostrar u ocultar el bloque dependiendo de si está o no visible, por lo que no hay que estar pendiente de cambiar su clase asociada.

Lo único que se ha tenido que realizar adicional, es una comprobación para poder alternar el nombre del botón entre expandir y ocultar.

Una vez conseguido este comportamiento, el siguiente paso es modificar la apariencia del listado.

Modificaciones en la apariencia del listado

En primer lugar, se ha añadido una comprobación para controlar grupos vacíos.

```
if (empty($gradespergroup))

$this->element('p', null, 'Todavía no hay puntuaciones.');
```

Todavía no hay puntuaciones.

Lo siguiente que se ha hecho es ordenar el listado, ya que no tenía sentido que no lo estuviera. Para ello, se ha modificado la consulta realizada en la función que obtiene la lista de usuarios junto con su puntuación, de manera que se le ha añadido el parámetro *order by grade desc*.

Al ser un ranking, lo lógico es que la lista tenga un número asociado, por lo que se ha cambiado el tipo de lista a una ordenada <*ol*>.

El siguiente paso ha sido añadir el avatar del usuario al lado del nombre, un detalle que además de mejorar la apariencia, servirá para identificar rápidamente a cada alumno.

Para ello, y ya que la clase Profile es la única que tiene una función para obtener el avatar, se ha tenido que utilizar la función staticGet para obtener todos los perfiles de los usuarios, y poder así usar su función *getAvatar*.

```
$profile = Profile::staticGet('nickname', $alumno);
$avatar = $profile->getAvatar(AVATAR_MINI_SIZE);
if ($avatar) {
$avatar = $avatar->displayUrl();
} else {
$avatar = Avatar::defaultImage(AVATAR_MINI_SIZE);
}
```

Como esta función obtiene la URL del avatar, únicamente falta añadir un elemento imagen dentro de cada elemento de la lista.

```
$this->element('img', array('src' => $avatar));
```

Por otro lado, y aprovechando que se han obtenido los perfiles de todos los usuarios, se ha modificado el nombre de los alumnos que se muestra, para que en vez de seleccionar el *nickname*, como se hacía anteriormente, se muestre el nombre completo, a través de la función *getBestName*.

Para distinguir el listado de profesores del de alumnos, se han creado dos funciones diferentes:

```
if ($this->user->hasRole('grader')) {
$this->showReportGrader();
}
else {
$this->showReportNoGrader();     }
```

El último cambio que se ha realizado es el formateado de las puntuaciones. Anteriormente, se mostraban con hasta 5 decimales, por lo que se ha modificado a través de la función de PHP *number_format*, para que solamente se muestren dos.

Con esto quedaría terminado el cambio de GradeReports, con una apariencia completamente diferente.

Añadir opción para exportar el listado de usuarios

Ficheros modificados: GradereportAction.php, grades.css, gradeoptionscsvAction.php, gradeexportcsvAction.php, GradesPlugin.php, gradecsvform.php

Un requisito para el nuevo Grade Reports era la posibilidad de exportar el listado de usuarios junto con sus puntuaciones a un fichero externo en formato CSV.

Tras valorar su viabilidad técnica, lo siguiente que se ha hecho es decidir el sitio óptimo para esta opción. Finalmente, se ha colocado al lado del botón "Expandir/Ocultar" vinculado a cada grupo.

Se ha añadido por tanto un nuevo enlace en la función *showReportGrader*, ya que esta opción sólo debe ser accedida por profesores.

Un detalle interesante, es que para evitar crear más formularios, con elementos *hidden* para pasar datos a la acción, se ha optado por utilizar un paso de parámetros de través de la URL (GET).

De esta manera, el ID del grupo, que es lo único que se necesita en la acción para generar el fichero CSV, se ha pasado como parámetro al crear la dirección destino del enlace.

```
$this->element('a', array('class' => 'grade-export-report', 'href' => common_local_url('gradeoptionscsv').'?group='.$group->id), 'Exportar CSV');
```

Del mismo modo que en el enlace "Expandir", se le ha aplicado una serie de reglas CSS para proporcionar al enlace una apariencia de botón.

Como se quería incluir degradados, se ha hecho uso de la herramienta online *GradientEditor* para generarlos.

http://www.colorzilla.com/gradient-editor/

El resultado final tras aplicar cambios de tamaño, de letra, colores… es el siguiente:

Una vez terminada la apariencia, es necesario crear la lógica de esta opción. Para ello, y ya que la URL apunta a la acción *gradeoptionscsv*, lo primero que hay que hacer es crearla. Esta acción va a ser la encargada de recibir el enlace anterior, y mostrar una nueva página en la que se pueda escoger el separador y el delimitador del fichero CSV a generar.

Como siempre, al añadir nuevas acciones y nuevos ficheros, es importante incluirlos en sus respectivos métodos, *onRouterInitialized* y *onAutoLoad*.

```
$m->connect('main/exportCSV/options', array('action' => 'gradeoptionscsv'));
```

```
case 'GradeoptionscsvAction':
include_once $dir . '/actions/' . $cls . '.php';
return false;
```

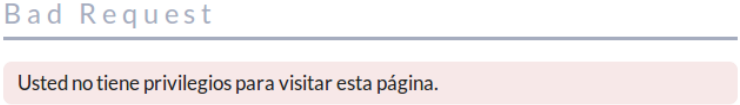
La página de selección debe contener un par de párrafos, un formulario que tenga dos cajas de texto para seleccionar el delimitador y el separador, y un botón *submit* para enviar los datos recogidos a una nueva acción.

Un detalle importante es que al utilizar GET, cualquier persona que obtenga este enlace, podrá acceder a la opción para exportar listados.

Para evitar esto, es importante lo primero de todo comprobar que el usuario actual es profesor, y de lo contrario, lanzar una excepción indicándolo.

```
if (!$this->user->hasRole('grader')) {
$this->clientError(_('Usted no tiene privilegios para visitar esta página.'));
return;
      }
```

De esta manera, si alguien que no tiene el rango *grader* consigue obtener la URL, se encontrará un error:



Al utilizar GET, también es posible que se modifique el parámetro manualmente en la URL, e incluso se pueda eliminar. Por ello, se ha añadido una comprobación que controle este aspecto:

```
if (empty($_GET['group'])) {
$this->showForm("Falta indicar el grupo asociado."); return;}
```

En este caso, para mostrar el mensaje de error de una manera elegante, se ha hecho uso de la función *showForm*:

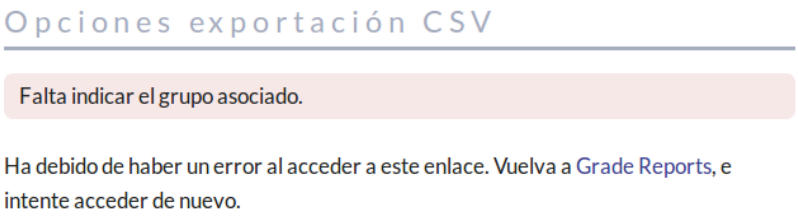
```
function showForm($error = null) {
$this->error = $error;
$this->showPage();
        }
```

Este método es preciso utilizarlo junto con *showPageNotice*, de manera que si hay un error, se muestre un elemento párrafo en la página indicándolo.

```
function showPageNotice() {
if ($this->error) {
$this->element('p', 'error', $this->error);
}
}
```

Por último, se ha añadido una comprobación de manera que si se produce un error, no se muestre el contenido normal de la página, ya que de lo contrario, se cargaría el formulario y se podrían realizar acciones con comportamientos inesperados.

```
else if ($this->error) {
$this->elementStart('p');
$this->raw('Ha debido de haber un error al acceder a este enlace. '
. 'Vuelva a <a href=' . common_local_url('gradereport') . '>Grade Reports</a>, e intente acceder de nuevo.');
$this->elementEnd('p');
}
```



Una vez controlados los posibles errores, el siguiente paso es mostrar el formulario de selección.
Para ello, se ha creado un nuevo formulario, *gradecsvform.php*. Este nuevo formulario debe tener una acción asociada, que sea la que se encargue de generar el fichero CSV. Esta nueva acción será *gradeexportcsv*.

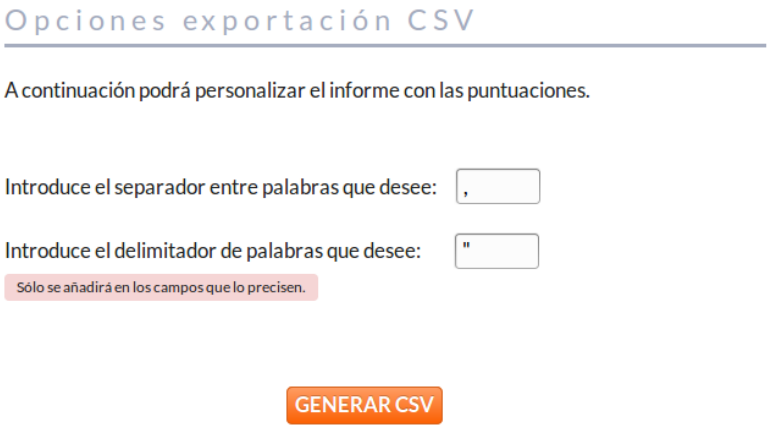
Un detalle importante, es que en los campos de texto para seleccionar el delimitador y el separador, se ha puesto un nuevo parámetro denominado *maxlength*, y se ha fijado a 1, de manera que no se puedan indicar separadores o delimitadores de más de un carácter.

Para no entrar en detalles excesivos, todas las reglas de apariencia que pertenecen a este formulario se pueden ver en el fichero *grades.css*.

Como este formulario necesita el ID del grupo, para pasárselo a la acción final que genere el fichero CSV, es necesario obtenerlo de la URL.

```
$groupid = $_GET['group'];
$formExport = new GradecsvForm($this, $groupid);
$formExport->show();
```

El resultado final del formulario es el siguiente:



El último paso es crear la acción que se encargue de generar el fichero CSV con el listado.

Para ello, creamos el fichero *GradeexportscsvAction.php*. En este caso, como esta nueva acción debe ejecutarse en segundo plano, y no se va a mostrar ninguna nueva página, ni se ha llamado ni se ha sobrescrito ningún método de showPage().

Por tanto, todas las acciones se realizarán en la función *handle* de la acción. El primer paso es obtener todos los campos del formulario, es decir, el *groupid* (elemento *hidden)*, el separador y el delimitador, todo ello utilizando la función *trimmed*.

Una vez obtenidos, se llama al método *getGradedNoticesAndUsersWithinGroup* para que devuelva la lista de usuarios junto con su puntuación para un grupo dado. Este es el mismo método que se utiliza en Grade Reports para generar los listados de cada grupo.

Por último, para estructurar mejor el código, se ha creado aparte la función encargada de generar el fichero CSV, y a la que se llama desde la función *handle*.

Los pasos que realiza esta nueva función son los siguientes. En primer lugar se deben colocar una serie de cabeceras para indicarle al navegador que se va a crear un fichero para descargar, y para especificar su codificación (UTF-8).

```
header('Content-Type: text/csv; charset=utf-8');
header('Content-Disposition: attachement; filename="' . $filename . '";');
```

En segundo lugar se debe abrir un fichero para escribir en él. Como en este caso el archivo va a ser temporal, es necesario indicarlo para que no lo cree físicamente.

```
$f = fopen('php://output', 'w');
```

Para generar el fichero CSV, se ha utilizado la función *fputcsv* siguiendo su manual, es decir, generando para cada línea un array con la información:

```
foreach ($array as $line) {
        fputcsv($f, $line, $separator, $delimiter);
}
```

Como el *array* que necesita esta función debe tener una estructura especial, es necesario procesar el *array* obtenido por la función getGradedNoticesAndUsersWithinGroup, y convertirlo en el formato esperado.

```
$arrayReport = Grades::getGradedNoticesAndUsersWithinGroup($groupid);
$arrayFinal = array();
foreach($arrayReport as $alumno => $puntuacion){
$arrayFinal[] = array($alumno, number_format($puntuacion,2));
        }
```

Por último, comentar algunos detalles interesantes de implementación.

- Se ha utilizado de la misma manera que en Grade Reports la función number_format para controlar el número de decimales de las puntuaciones (2).
- Se han añadido dos comprobaciones de manera que si los parámetros recibidos (separador y delimitador) son vacíos, se restablezcan los de por defecto.
- Es importante conocer que el separador no se incluye siempre en el fichero CSV, solamente cuando la función *fputcsv* lo considera necesario.

El proceso de generación del fichero es similar al que se muestra a continuación:


```
alumno2,2.00
alumno1,2.00
borja,1.50
alvaro,1.50
```

Grade reports no se visualiza correctamente al añadir el botón para Exportar CSV

Ficheros modificados: grades.css, GradereportsAction.php

Al añadir el nuevo enlace para exportar los listados a un fichero CSV, los nombres largos de asignaturas descuadran la apariencia del Grade Reports.

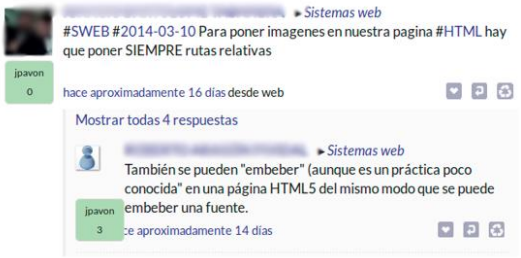
Para solucionar esto, lo que se ha hecho es cambiar la creación de los <h3> en *GradereportsAction*, para que se creen con un "title", en el que se muestre el nombre entero al colocar el ratón encima.

Por otro lado, a través de reglas CSS se ha fijado un tamaño máximo al nombre, de manera que si la asignatura lo supera, se acorte y se muestren puntos suspensivos.

El recuadro con las puntuaciones en respuestas no se muestra correctamente

Ficheros modificados: grades.css, GradesPlugin.php

Debido al reducido tamaño de las respuestas en BoloTweet, las reglas CSS que se aplican al recuadro con las puntuaciones provocan que se visualice de forma incorrecta.



Ya que no es posible disminuir tanto la letra como para que quepa en un recuadro de menor tamaño, se ha optado por realizar un comportamiento diferente.

Para poder distinguir a nivel de implementación entre noticia normal, o respuesta, se ha tenido que depurar para observar si existía algún tipo de atributo que las diferenciara. Efectivamente, las noticias normales son de la clase: *ThreadedNoticeListItem*, mientras que las respuestas pertenecen a *ThreadedNoticeListSubItem*.

Por ello, se ha hecho la siguiente comprobación.

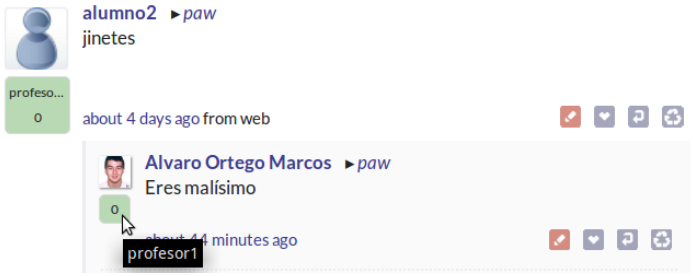
```
if (get_class($args) === 'ThreadedNoticeListSubItem') {
                    $args->out->raw($gradeValue);
                } else {
                    $args->out->raw($grader . '<br/>' . $gradeValue);
                }
```

Lo que se ha hecho es que si la noticia es de tipo respuesta, únicamente se muestre la nota, mientras que si no lo es, se muestre como siempre, profesor más nota.

Además, al recuadro con la puntuación se le ha añadido un campo *title* en el que se muestra el nombre del *grader*, de manera que si es respuesta, colocando el ratón encima, se pueda ver qué profesor ha realizado la puntuación.

Por último, solamente queda cambiar la apariencia del recuadro en las respuestas a través de CSS.

Se ha modificado el tamaño (20px), el *padding*, y la posición *top*, dando como resultado lo siguiente:



Añadir un enlace en Grade Reports para cada alumno

Ficheros modificados: grades.css, GradereportAction.php, GradeshowuserAction.php, GradesPlugin.php, Grades.php

Una de las nuevas opciones del *Grade Reports* es la posibilidad de acceder a los mensajes de un determinado alumno en un grupo concreto de manera rápida.

Para lograr este comportamiento, la idea es generar un enlace vinculado a cada alumno, de manera que al pulsarlo, se muestre una página con toda la recopilación de *tweets* puntuados de ese alumno en ese grupo.

Al no existir ninguna acción predefinida que haga esto, se ha tenido que desarrollar una acción completa manualmente, que recopile los mensajes asociados a un alumno, y que se encargue de mostrarlos divididos en páginas.

Esta es la primera vez que se trabaja con listados de noticias, y con su visualización, por lo que se tratará de estudiar otras acciones similares para poder recrear su comportamiento.

Lo primero que se ha hecho es decidir qué URL asignarle a esta acción. Tras valorar distintas posibilidades, se ha optado por una URL dinámica, del tipo "/gradereport/grupo/usuario", que se modifique dependiendo del alumno y del grupo al que se acceda.

La creación de URL dinámicas en StatusNet es un tema complejo por lo que se tratará de explicar a continuación de la forma más sencilla posible.

Para poder utilizar una dirección dinámica con la estructura mostrada anteriormente, se ha tenido que incluir el siguiente código:

```
$m->connect('main/gradereport/:nickgroup/:nickname',
array('action' => 'gradeshowuser'),
array('nickgroup' => Nickname::DISPLAY_FMT,'nickname' => Nickname::DISPLAY_FMT)
);
```

Como se puede ver, en la URL se colocan dos variables (con dos puntos delante del nombre), en este caso *nickgroup* y *nickname*. Lo siguiente que es necesario es indicarle a la acción qué tipo de variables son.

En este caso, se ha utilizado el tipo *Nickname::DISPLAY_FMT*, que no es más que un patrón.

```
const DISPLAY_FMT = '[0-9ª-zA-Z_]{1,64}';
```

Una vez definido esto, se explicará el funcionamiento interno de las URLs dinámicas. Al acceder a un enlace, en la función *main* de StatusNet se realiza una búsqueda de acciones vinculadas a esa URL. Cuando las URL son dinámicas, se realiza una comprobación a través de expresiones regulares, de manera que si la forma de la *url* coincide con la de una expresión regular previamente almacenada, se escoge su acción asociada.

En este caso, la URL dinámica tiene la siguiente forma.

```
main/gradereport/:nickgroup/:nickname
```

Si se accede de manera interna al valor de esta URL, el resultado es el siguiente:

```
"#^main/gradereport/(?P<nickgroup>[0-9ª-zA-Z_]{1,64})/(?P<nickname>[0-9ª-zA-
Z_]{1,64})$#"
```

Como se puede ver, no es más que una URL en la que se han reemplazado las variables por sus tipos. De esta manera, cuando se acceda a un usuario desde el *Grade Reports*, y se genere una url del tipo:

"main/gradereport/grupo/usuario"

Coincidirá con el valor interno de la URL dinámica, y por tanto se ejecutará la acción asociada, en este caso, *gradeshowuser*.

Una vez entendido el funcionamiento básico de las URL dinámicas, lo que se ha hecho es modificar el comportamiento del Grade Reports, de manera que al generar la lista de alumnos, en vez de crear sus nombres como texto plano, se sustituye por un enlace a la nueva acción creada.

```
$this->elementStart('a', array('class' => 'user-link-report', 'href' =>
common_local_url('gradeshowuser', array('nickgroup' => $group->nickname, 'nickname'
=> $profile->nickname))));
```

Por otro lado, comentar que se han creado algunas reglas CSS para evitar que los enlaces se subrayen, y para separarlos del avatar.

El siguiente y último paso es crear toda la lógica de la nueva acción, por lo que en primer lugar se debe crear el fichero apropiado, *GradeshowuserAction.php*, e incluirlo en el método *onAutoLoad* de GradesPlugin.

El comportamiento final de esta acción es el siguiente. Tras pulsar en el nombre de un alumno desde un listado de GradeReports, se deberá redirigir al usuario a una página en la que se muestren los mensajes puntuados del alumno en el grupo que corresponda. Para ello, va a ser necesario en primer lugar obtener esos mensajes, y por último mostrarlos en la página.

Sin embargo, deberá controlarse la visualización y apariencia de los mismos, y lo que es más complejo, el mecanismo de división de mensajes en páginas.

Tras realizar una gran cantidad de pruebas para entender cómo realizaban todo este proceso acciones similares como *showstream*, que muestra los mensajes de un usuario, se ha conseguido recrear este funcionamiento con un código relativamente sencillo.

Lo primero que es necesario es incluir en la acción un atributo "page" que se encargue de almacenar y de controlar la navegación entre páginas.

Por otro lado, en la función *prepare*, es necesario obtener el nombre del usuario y el nombre del grupo para localizar los mensajes. Por ello, y ya no se ha utilizado ningún formulario para el paso de variables, es necesario coger los valores de la URL.

```
$alumno = $this->trimmed('nickname');
$group = $this->trimmed('nickgroup');
```

El siguiente paso que se debe realizar es una comprobación para conocer qué página se debe mostrar.

```
$this->page = ($this->arg('page')) ? ($this->arg('page') + 0) : 1;
```

Una vez hecho esto, el último paso es obtener las noticias del alumno y mostrarlas. Sin embargo, todas las acciones predefinidas que trabajan con listas de noticias, como *showstream*, *all*, *reply*, *inbox*… disponen de clases personalizadas, con atributos propios y con funciones exclusivas que se encargan de coger la lista de mensajes que corresponda en cada caso.

Como para esta nueva acción no hay nada disponible, ni siquiera se puede reutilizar un *stream* ya creado, ya que si se utiliza por ejemplo *ProfileNoticeStream*, al llamar a la función getNotices, se van a obtener las noticias que se deberían mostrar en la acción *showstream*, ya que está programado para ello, mientras que en este caso esos no son los mensajes que se deben mostrar.

Por ello, lo que se ha hecho es estudiar el funcionamiento de todas estas funciones, ver sus partes en común, las partes necesarias, e intentar reconstruir el mismo comportamiento para la nueva acción.

Las cosas imprescindibles que se realizan en todas estas acciones son las siguientes:

En todos los casos se ejecuta una función *getNotices* que en algún punto de su ejecución termina llamando a alguna función que devuelve los *ids* de las noticias relevantes para esa acción (*getNoticeIDs)*.

Posteriormente, a través del método *MultiGet* de *Notice* y junto con los ids recibidos, se obtienen todas las noticias relevantes de la base de datos.

Una vez reunidas las noticias, utilizan métodos propios para filtrarlas, y a continuación las guardan en un ArrayWrapper a partir del cual crean un *NoticeList* final que será el que se muestre.

Comprendido el funcionamiento de estas acciones, tan sólo queda recrearlo en la manera en que sea necesario.

El primer paso ha sido desarrollar una consulta SQL que seleccione los ID's de las noticias que correspondan, emulando el *getNoticeIDs* del resto de acciones. Como se deben seleccionar los mensajes de un determinado alumno en un determinado grupo, y que además estén puntuados, la consulta precisa del *userid* y del *groupid*.

Se ha creado por tanto la función *getNoticeFromUserInGroup*, que como se puede observar, lleva el parámetro *distinct* para evitar repeticiones de noticias ocasionadas por las puntuaciones múltiples.

```
$qry = 'select distinct noticeid '
                . 'from grades g, group_inbox gi, notice n '
                . 'where g.noticeid = gi.notice_id '
                . 'and gi.group_id = ' . $groupid
                . ' and g.noticeid = n.id '
                . ' and n.profile_id = ' . $userid
                . ' order by n.created desc';
```

Por otro lado, se debería crear la función *getNotices* que filtrara las noticias y generara el *ArrayWrapper*. Sin embargo, como las noticias se han seleccionado manualmente a través de esta consulta SQL, el filtrado no es necesario, por lo que únicamente se debe generar el ArrayWrapper de noticias a partir de los ID obtenidos.

Para implementar esta función se han realizado varias opciones. La primera implementación utilizaba el mismo mecanismo que el resto de acciones de este tipo, es decir, se obtienen los IDs de las noticias que correspondan, se obtienen todas con la función Notice::*MultiGet*, y posteriormente se almacenan en el *ArrayWrapper* únicamente las que se vayan a mostrar dependiendo de la página.

Sin embargo, este comportamiento no es óptimo, pues no tiene sentido obtener 1000 noticias, para posteriormente mostrar 20. Por ello, se ha *re-implementado* el comportamiento de manera que ahora, primero se seleccionan los ID's de las noticias que se van a mostrar en la página, y únicamente serán esas las que se obtengan utilizando Notice::MultiGet y las que se almacenen en el *ArrayWrapper*.

```
$total = $offset + $limit;

        for ($i = $offset; $i < $total; $i++) {
            $idsFinal[] = $ids[$i];
        }

        $notices = Notice::multiGet('id', $idsFinal);
```

Un detalle interesante es que para que la función getNotices sepa qué noticias son las que se deben mostrar en la página actual, es necesario llamarla de la siguiente forma:

```
$ids = Grades::getNoticeFromUserInGroup($this->alumno->id, $this->group->id);
$this->notice = $this->getNotices(($this->page - 1) * NOTICES_PER_PAGE, NOTICES_PER_PAGE + 1, $ids);
```

Por último, solamente queda generar el *NoticeList* a partir del *ArrayWrapper* creado anteriormente y mostrarlo:

```
$nl = new NoticeList($this->notice, $this);
$cnt = $nl->show();
```

Para generar los enlaces de "Anterior" y "Siguiente", en el caso de que exista más de una página, se utiliza el método *pagination* de la clase Action de la siguiente manera:

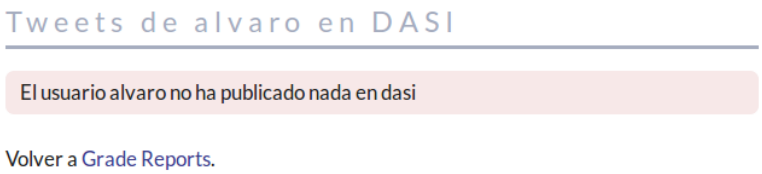
```
$this->pagination($this->page > 1, $cnt > NOTICES_PER_PAGE, $this->page, 'gradeshowuser', array('nickgroup' => $this->group->nickname, 'nickname' => $this->alumno->nickname));
```

Con esto, se habría conseguido recrear un funcionamiento similar al de las otras acciones de una manera mucho más sencilla, y con un resultado completamente igual.

Además, se ha añadido en la función *getNotices* una comprobación, de manera que si el usuario aún no tiene ningún *tweet* puntuado, se muestre un mensaje indicándolo.

La forma de mostrar este mensaje de error se explicó anteriormente en la tarea *Añadir opción para exportar el listado de usuarios*, utilizando las funciones *showForm* y *showPageNotice*.

El resultado en este caso es el siguiente:



Como se puede ver en la imagen anterior, esta acción muestra un título antes de la lista de mensajes, del tipo:

Tweets de Usuario en Grupo

Para hacer más completa esta acción, y recrear el comportamiento también de otras opciones, se ha implementado el título de manera dinámica, de manera que dependiendo de la página en la que se encuentre, el título varíe.

Para ello, hay que sobrescribir la función *title*, e incluir una comprobación de manera que si la página no es la primera, se añada al título la página actual.

```
if ($this->page == 1) {

return sprintf(_('Tweets de %s en %s'), $this->alumno->nickname, strtoupper($this->group->nickname));

} else {

return sprintf(_('Tweets de %1$s en %2$s (%3$d)'), $this->alumno->nickname, strtoupper($this->group->nickname), $this->page);

      }
```

Hasta ahora el resultado conseguido es el siguiente:

Por último, y para completar más la acción proporcionándola más funcionalidad, se han realizado dos cambios más.

El primero de ellos ha sido el de sustituir en la acción, el perfil del usuario actual del *aside* derecho por el perfil del alumno que se esté visitando.

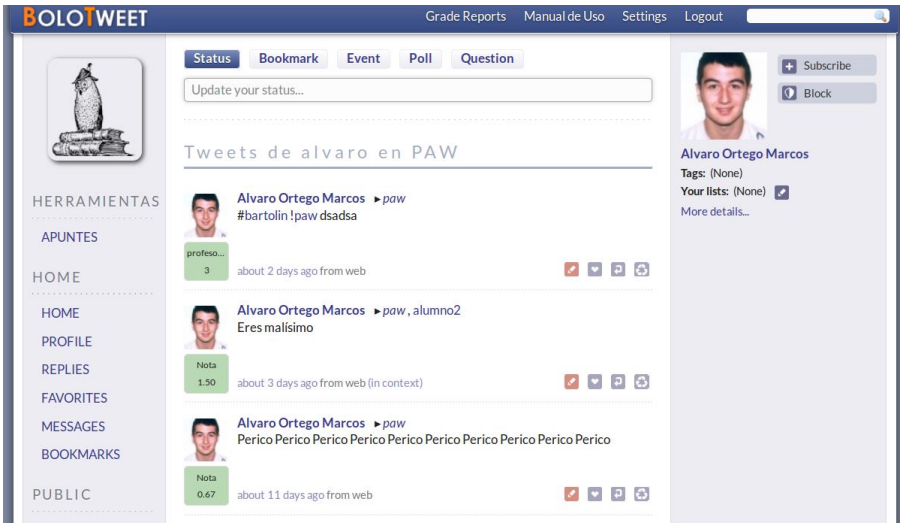
Cuando se utiliza la función showContent para sobrescribir el contenido de la página, por defecto en el aside derecho se muestra el perfil del usuario actual, tal y como se puede ver en la imagen actual.

Esta información no es relevante ni aporta nada de información a la acción en sí, por lo que ha decidido añadirse la información del usuario que se está visitando.

Para lograr esto, es necesario sobrescribir el método *showProfileBlock*, y crear el nuevo bloque con el perfil del alumno.

```
$block = new AccountProfileBlock($this, $this->alumno);

$block->show();
```

Sólo con esto código se consigue sustituir esa parte, añadiendo en este caso la información del alumno en cuestión.



El segundo y último cambio, ha sido la creación de un bloque con estadísticas en ese mismo *aside*, para proporcionar al profesor más información.

Para la creación de este bloque, se debe sobrescribir el método *showSections()*. Todos los elementos que se incluyan esta función se mostrarán debajo del perfil del alumno, pudiendo crear tantas secciones como se quiera.

En este caso, se ha decidido colocar una única sección con estadísticas, en la que se han incluido los siguientes aspectos:

- Número de tweets de ese usuario en ese grupo.
- Número de tweets de esos, puntuados.
- Puntos totales
- Puntuación media

Para poder calcular todos esos valores, se ha creado una función aparte, *showStatistics*, que es la que se va a encargar de rellenar la nueva sección.

Dentro de este nuevo bloque, para cada elemento se ha implementado un esquema similar al siguiente:

```
$this->elementStart('p');
$this->raw('Número de Tweets: ');
$this->elementStart('span', array('class' => 'statistics-span'));
$this->raw($this->numeroTweets);
$this->elementEnd('span');
$this->elementEnd('p');
```

En este esquema lo que se hace es crear en primer lugar el párrafo donde se incluirá el nombre del elemento y su valor.

Tal y como se puede observar, para acceder al valor se utiliza un atributo de clase, en este caso, *numeroTweets*. Estos atributos se rellenan a través de la función *generarEstadisticas*, que se ejecuta en la función *prepare*.

- Para poder conocer el valor del número de tweets totales de un usuario en un grupo, se ha tenido que crear la función getNumberTweetsOfUserInGroup y la correspondiente consulta:

```
$qry = 'select count(gi.notice_id) as number'
. ' from group_inbox gi, notice n '
. 'where gi.notice_id = n.id '
. 'and gi.group_id = ' . $groupid
. ' and n.profile_id = ' . $userid;
```

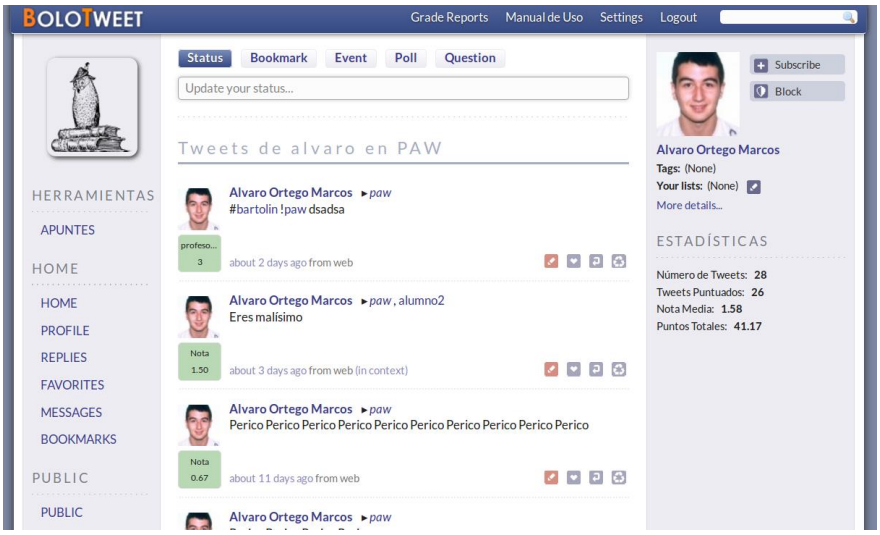
- Para el número de tweets puntuados, se ha utilizado la función existente *getNoticeFromUserInGroup*, con la única diferencia de que a la respuesta obtenida, se la aplica la función *count* para poder obtener el número de mensajes.

- Para obtener la nota media, y los puntos totales, se ha desarrollado una nueva función, *getNotaMediaYTotalofUserinGroup*, con una consulta bastante más compleja en la que se consiguen devolver ambos valores.

```
$qry = 'select avg(tmp.gradeAvg) as media, sum(tmp.gradeAvg) as total '
. 'from (select avg(g.grade) as gradeAvg '
. 'from grades g, group_inbox gr, notice n '
. 'where g.noticeid = gr.notice_id '
. 'and gr.group_id = ' . $groupid
. ' and g.noticeid = n.id '
. 'and n.profile_id = ' . $userid
. ' group by g.noticeid) as tmp';
```

Una vez obtenidos todos los valores, se les ha aplicado la función *number*_format para fijar el número de decimales. Por último, se han aplicado algunas reglas CSS para mejorar la apariencia de esta sección.

Con todo esto se ha conseguido una acción totalmente funcional, y que tiene el siguiente aspecto final:



Grade Reports solamente muestra los alumnos puntuados

Ficheros modificados: Grades.php, GradereportAction.php

Tal y como se había planteado el *Grade Reports*, únicamente se obtenían de la base de datos los alumnos con puntuaciones. Este comportamiento no es del todo correcto, pues si se desea exportar una lista con las calificaciones, va a haber una serie de alumnos que sí pertenecen a ese grupo, y que no se van a exportar.

Para solucionar esto, lo que se ha hecho es crear una función en Grades que se llama *getMembersNicksExcludeGradersAndAdmin*, que se encarga de devolver todos los miembros de un grupo, excluyendo a los profesores y administradores.

Por otro lado, como en la función que se utiliza para generar el listado, *getGradedNoticesAndUsersWithinGroup*, únicamente se devuelven los usuarios con puntuaciones, lo que se ha hecho es ir comparando el *array* de todos los miembros con el *array* de los puntuados, de manera que los alumnos que no estuvieran en el segundo, se les ha ido añadiendo al final con una puntuación de 0.

Es importante el detalle de añadirlos al final del *array*, ya que este está ordenado por orden de puntuaciones.

El código que realiza esto es el siguiente:

```
$gradespergroup = Grades::getGradedNoticesAndUsersWithinGroup($group->id);
$nicksMembers = Grades::getMembersNicksExcludeGradersAndAdmin($group->id);

            foreach ($nicksMembers as $nick) {
                if (!array_key_exists($nick, $gradespergroup)) {
$gradespergroup[$nick] = '0';
                }
            }
```

De esta manera, cuando se recorra el *array* con puntuados para generar el listado, ahora sí se mostrarán todos los miembros del grupo, tengo o no mensajes puntuados.

Al exportar un listado de alumnos no se incluyen los no puntuados

Ficheros modificados: GradeexporcsvtAction.php

En este caso, al generar el fichero CSV con los alumnos de un grupo, y al utilizarse la misma función que al generar los listados del *Grade Reports*, sucede exactamente lo mismo que en la tarea anterior, y es que únicamente se exportan los usuarios con puntuaciones.

Para solucionarlo, se ha realizado exactamente el mismo proceso que en la tarea anterior, de manera que tras obtener los miembros totales del grupo, se ha realizado un filtrado de los ya puntuados, y el resto se ha añadido al array con una puntuación de 0.

De esta manera, cuando se genere la exportación los ficheros contendrán información de todos los alumnos pertenecientes a un grupo, tengan o no puntuaciones.

Vincular profesor automáticamente al crear un grupo

Ficheros modificados: GradesPlugin.php

Como en una de las modificaciones de *BoloTweet* se ha restringido la creación de grupos únicamente a profesores, tiene sentido que al crear un grupo, se le vincule automáticamente con él, sin necesidad de solicitárselo al administrador de BoloTweet.

Para lograr este comportamiento, se ha tenido que utilizar el hook *EndGroupSave*, ya que se ejecuta únicamente tras la creación de nuevos grupos.

Se ha añadido por tanto en *GradesPlugin.php* una función que responda a este evento, y se ha incluido en su interior el siguiente código:

```
function onEndGroupSave($group) {
        $user = common_current_user();
        if (!empty($user)) {
            if ($user->hasRole('grader')) {
                Gradesgroup::vincularGrupo($user->id, $group->id);
}
      }
   }
```

De esta manera, cuando un usuario con rango *grader* cree de manera gráfica un grupo, además de aparecer como administrador, se le vinculará automáticamente como profesor del mismo, lo que le permitirá poder puntuar cualquier mensaje de su interior.

Ofrecer la posiblidad de añadir graders a un grupo de manera gráfica

Ficheros modificados: GradesPlugin.php, MakegraderAction.php, MakegraderForm.php, grades.css, Gradesgroup.php, groupmembers.php

Hasta ahora, cuando se quería que un usuario se convirtiera en *grader* de un grupo, se debía hacer a través de scripts en línea de comandos, y únicamente podía hacerlo el administrador del sistema.

Se ha visto la necesidad de poder proporcionarle al profesor creador del grupo, el privilegio de poder nombrar ayudantes de manera gráfica, sin necesidad de contactar con el administrador de *BoloTweet*.

Es importante subrayar que únicamente podrá crear profesores en un grupo el creador del mismo.

Tras barajar diversas opciones sobre dónde incluir esta nueva opción, se ha decidido colocarla junto con el resto de opciones vinculadas a cada miembro del grupo.

Para ello, se ha hecho uso del *hook onEndProfileListItemActionElements*, el cual permite añadir elemento al final de esa lista de opciones.

Tras comprender el funcionamiento del resto de opciones ya existentes, ha sido necesario crear un formulario junto con su acción asociada.

Se ha creado por tanto los ficheros *MakegraderAction.php* y *MakegraderForm.php.*

La acción se encargará de añadirle los rangos *grader* y *deleter* al usuario en caso de que no los tuviera, y por último vincularlos al grupo como profesores.

```
if (!$this->profile->hasRole('grader')) {
            $this->profile->grantRole('grader');
   }
   if (!$this->profile->hasRole('deleter')) {
            $this->profile->grantRole('deleter');
   }

   $result = Gradesgroup::vincularGrupo($this->profile->id, $this->group->id);
```

Por otro lado, el formulario simplemente dispondrá de un único botón *submit*, que será el elemento que se mostrará, y varios elementos *hidden* en los que pasar datos a la acción.

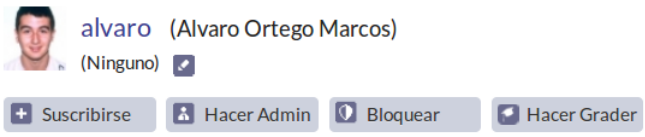
Una vez creados ambos archivos, y como en el resto de ocasiones, es necesario añadir un nuevo elemento en la función *onRouterInitialized*, y actualizar con los nuevos ficheros la función *onAutoLoad*.

Una vez creada la acción y el formulario, el siguiente paso ha sido controlar en qué miembros se debía mostrar la opción para convertir a *grader*. Como no basta con comprobar si un usuario ya dispone de ese rango, ya que puede tenerlo pero estar vinculado a otro grupo, y por tanto si se debería mostrar la opción, se ha tenido que crear en *Gradesgroup.php* una nueva función, *isGrader*, cuyo objetivo es responder si un usuario está vinculado como profesor a un determinado grupo.

```
$qry = 'SELECT gg.userid'
                . ' FROM grades_group gg'
                . ' where gg.userid=' . $userid
                . ' and gg.groupid =' . $groupid;
```

Por último, se ha diseñado con *Adobe Photoshop* un icono para mostrarse junto al botón. Además, se han aplicado una serie de reglas CSS para que las opciones relativas a cada usuario se muestren en horizontal, ya que de lo contrario, con la nueva opción separaban demasiado un usuario del siguiente.

El resultado de todo esto es el siguiente:



Sin embargo, cuando el usuario no es Administrador del grupo, es decir, cuando únicamente tiene como opción para cada usuario "*Suscribirse*", no tiene sentido que se muestren las opciones en horizontal, ya que se desperdicia espacio.

Por tanto se ha decidido modificar el comportamiento dependiendo del usuario actual. Para poder distinguir entre uno y otro, se ha modificado el fichero *groupmembers.php*, encargado de generar la lista de miembros de un determinado grupo, y se ha realizado lo siguiente:

```
if ($user->isAdmin($this->group)) {
      $this->elementStart('div', array('class' => 'member-list-admin'));
}
```

La idea es, que si el usuario actual es administrador, se genere un bloque *div* por encima, con una clase específica. Por otro lado, se han modificado las reglas CSS para colocar las opciones en horizontal, de manera que ahora únicamente se apliquen cuando exista un *div* con la clase *.member-list-admin* por encima, y así conseguir un comportamiento u otro dependiendo del usuario actual.

**Desde móviles sale un error con la etiqueta
**

Ficheros modificados: GradesPlugin.php

Al acceder desde dispositivos móviles a *BoloTweet*, se obtenía un error generado por una etiqueta <*br*>. Finalmente, este mensaje era provocado por la etiqueta de salto que se aplica en el recuadro con las puntuaciones, para poner en una línea distinta el nombre del profesor y la nota.

Para solucionarlo, simplemente ha habido que modificar la etiqueta <*br*> por <*br/*>.

NotesPDF:

La idea de este plugin es ofrecer al alumno la posibilidad de obtener a partir de las ideas escritas por ellos, o por sus compañeros, una serie de apuntes que sirvan como herramienta de ayuda al estudio para enfrentarse a las asignaturas.

Los apuntes se generan siempre para un grupo dado, es decir, son individuales para cada asignatura.

Tal y como se ha planteado, NotesPDF ofrece dos tipos principales de apuntes, automáticos y personalizados.

Los apuntes automáticos son una recopilación de las mejores ideas de un determinado grupo. Para conocer cuáles son las mejores ideas, se utilizarán las calificaciones proporcionadas por los profesores, de manera que en este caso, se seleccionarán aquellas que tengan una puntuación de 2 o 3.

Por otro lado, los apuntes personalizados proporcionan la posibilidad de generar una serie de apuntes de acuerdo a las necesidades de cada usuario. De esta manera, el usuario podrá personalizar 3 parámetros de sus apuntes.

El primero de ellos es el ***hashtag***, pudiendo elegir entre "Todos", o bien uno en concreto.

El segundo parámetro es el ***autor***, de manera que puede seleccionar o bien todos los autores, o bien todas las ideas de un alumno concreto.

Por último, el tercer parámetro es la ***puntuación***, seleccionando los tweets calificados con todas las puntuaciones, o bien con una en concreto.

Los apuntes generados contendrán información de la asignatura, de la fecha de generación, y además incluirán una sección autores en la que se podrán los usuarios que han realizado cada idea.

Crear la estructura interna del plugin

Ficheros modificados: NotesPDFPlugin.php, notes.css, notes.js

Para poder añadir posteriormente toda la lógica del Plugin, es necesaria crear la estructura básica.

Se han creado las siguientes carpetas:

- actions
- classes
- css
- js
- lib
- scripts

Añadir un submenú en el lateral de la página, con un enlace a Apuntes

Ficheros modificados: NotesPDFPlugin.php

Para añadir un enlace en el *aside* izquierda se ha utilizado el *hookstartDefaultLocalNav*.

Una vez creada la función vinculada a ese evento en *NotesPDFPlugin.php*, se ha mirado la implementación del resto de enlaces del *aside* para realizarlo de la misma manera.

El código final es el siguiente:

```
$actionName = $action->trimmed('action')
    $user = common_current_user();
        if (!empty($user)) {
            $action->out->elementStart('li');
            $action->out->element('h3', null, 'Herramientas');
            $action->out->elementStart('ul', array('class' => 'nav'));
            $action->menuItem(common_local_url('notesgroups'),       _m('Apuntes'),
            _m('Apuntes en PDF'), ($actionName === 'notesgroups' || $actionName
            === 'notescustomize'), 'nav_notespdf');
            $action->out->elementEnd('ul');
            $action->out->elementEnd('li');
}
```

Escoger qué librería usar para generar apuntes.

Ficheros modificados: <Ninguno>

Uno de los pasos más importantes de este plugin es una buena elección de la librería que se va a utilizar para generar los ficheros PDF.

Tras barajar diferentes opciones, y valorar los pros y los contras de cada una, finalmente se ha decidido utilizar la librería FPDF en su versión **v1.7**.

http://www.fpdf.org/

Es una librería versátil, sencilla y actualizada, por lo que en un principio no debería dar ningún problema.

Para poder utilizar esta librería, es necesario descargar todos sus archivos, incluirlos en este caso en la carpeta *lib*, y posteriormente se deberá importar el archivo principal de la librería desde la acción que lo necesite.

Crear acción para seleccionar el grupo del que generar apuntes

Ficheros modificados: NotesgroupsAction.php, NotesPDFPlugin.php, Notesgroupsform.php, notes.css

Una vez se pulsa en el enlace creado en el *aside* izquierdo, se debe redirigir a una acción que se encargue de mostrar una primera página en la que el usuario pueda seleccionar para qué grupo desea crear apuntes.

Lo primero que se ha hecho es crear el fichero *NotesgroupsAction.php*, donde se implementará toda esta lógica.

La idea de esta acción es que se obtengan los grupos a los que pertenece el usuario actual, y posteriormente se muestre para cada uno de ellos un formulario que permita acceder a la siguiente acción, donde se seleccionará el tipo de apuntes.

Sin embargo, para no caer en comportamiento extraños, es importante que en este primer paso se filtren los grupos con puntuaciones, de manera que aquellos que no tengan ningún mensaje puntuado, no dispondrán del formulario activo para pasar a la siguiente acción.

El primer paso por tanto ha sido obtener los grupos del usuario actual. Se ha añadido una comprobación de manera que se muestre un mensaje si el usuario no está unido aún a ningún grupo.

```
$groupsUser = $this->user->getGroups()->fetchAll();
      if (empty($groupsUser)) {
             $this->elementStart('p', array('class' => 'notes-empty-groups'));
             $this->raw("Todavía no perteneces a ningún grupo.");
             $this->element('br');
             $this->raw("Prueba uniéndote a alguno de los ");$this->element('a',
             array('href' => common_root_url() . 'groups/'), "Grupos Disponibles");
             $this->elementEnd('p');
       }
```

Para comprobar si un grupo tiene o no puntuaciones, se ha utilizado la función de Grades, *getIDsGroupsWithGrades*. De esta manera, lo que se hace es recorrer los grupos a los que pertenece el usuario, y para cada uno de ellos se comprueba si existe en el *array* devuelto por la función de Grades.

Para poder escoger el tipo de apuntes en el caso de que un grupo tenga puntuaciones, es necesario crear un formulario que enlace con la siguiente acción. Por ello, se ha creado el formulario *Notesgroupsform*, en el que únicamente se ha incluido un elemento *hidden* para pasar el *groupid* a la próxima acción, y por otro lado el botón de tipo *submit* para enviar el formulario.

Un detalle importante, es que para poder crear un formulario desactivado en el caso de los grupos sin puntuaciones, se ha realizado a través de un parámetro adicional *disabled*, de manera que cuando este parámetro está a *true*, el botón *submit* se crea desactivado.

```
        if ($this->disabled == 'true') {
                    $this->out->element('input', array('type' => 'submit',
                        'id' => 'notes-submit-' . $this->idGroup,
        'class' => 'submit',
                        'value' => _('Generar Apuntes'),
                        'title' => _('Genera apuntes de este grupo en PDF'),
                        'disabled' => $this->disabled));
        }
```

Una vez creado el formulario, en la acción solamente queda es hacer una cosa u otra dependiendo de si el grupo tiene o no, mensajes puntuados.

Si el grupo aún no tiene calificaciones, entonces se muestra el formulario desactivado, y por otro lado se genera un párrafo indicándolo.

Si por el contrario el grupo si tiene alguna puntuación, simplemente se crea el formulario y se muestra.

```
if (!in_array($group->id, $groupsGraded)) {
        $butGenerate = new Notesgroupsform($this, $group->id, 'true');
        $butGenerate->show();

        $this->elementStart('div');
        $this->elementStart('p', array('class' => 'notes-error-group-text'));
        $this->raw("No es posible generar apuntes. Grupo sin puntuaciones.");
        $this->elementEnd('p');
        $path = common_path('local/plugins/NotesPDF/css/x.gif');
        $this->element('img', array('class' => 'notes-error-group', 'src' => $path));
        $this->elementEnd('div');
} else {
        $butGenerate = new Notesgroupsform($this, $group->id);
        $butGenerate->show();
}
```

Un detalle interesante es que para adornar la lista de grupos, además de reglas CSS, se ha utilizado la función getAvatar de Group, para obtener y mostrar junto a cada grupo de la lista su avatar correspondiente.

```
$avatar = $group->getAvatar();
$this->element('img', array('src' => $avatar, 'width' => '48', 'height' => '48'));
```

El resultado tras crear todo este comportamiento es el siguiente:

Crear acción para seleccionar tipo de apuntes

Ficheros modificados: NotescustomizeAction.php, NotesPDFPlugin.php, NotesPDF.php, notes.css, Notescustomizeform.php

Una vez el usuario accede a generar apuntes para un grupo determinado, el siguiente paso debe ser una página en la que pueda seleccionar el tipo de apuntes que desea.

Por ello, lo primero que se ha hecho es crear la acción correspondiente, *NotescustomizeAction*.
Esta acción no debe tiene ninguna lógica compleja, tan sólo debe mostrar una página con un poco de información, y crear el formulario con ambas opciones, apuntes personalizados, y apuntes automáticos, que posteriormente enlazará con la acción final que genere el archivo PDF.

```
$this->element('h2', null, 'Apuntes para el grupo ' . $group->getBestName());

            $this->elementStart('p');
$this->raw('A continuación personalice los apuntes.');
$this->elementEnd('p');

            $optionsForm = new Notescustomizeform($this, $group->id);
$optionsForm->show();
```

A continuación por tanto, se ha creado el fichero correspondiente al formulario, *Notescustomizeform*.
En este caso, es el formulario el que tiene más complejidad, pues debe incluir dos secciones completamente distintas, una para cada tipo de apuntes.

La sección para apuntes automáticos contendrá varios elementos de tipo párrafo, así como un botón *submit*.

Por otro lado, la sección dedicada a apuntes personalizados contendrá de nuevo elementos párrafo, un botón *submit*, y en este caso 3 elementos de tipo *select (combobox*), cada uno correspondiente a los parámetros que el usuario puede personalizar.

Algunos aspectos importantes en la implementación de esta parte:

- Al existir más de un botón *submit* en el mismo formulario, cada uno de ellos tiene un valor interno diferente, de manera que en la próxima acción se pueda distinguir qué opción se ha escogido.
- Para poder cargar los elementos de tipo *select* con las opciones que correspondan, se han tenido que crear funciones y consultas que obtengan la información correspondiente en cada caso.

Para rellenar el *combobox* con los *hashtag* existentes, se ha creado la función *getTagsOfUserWithGradeInGroup*.

```
$qry = 'select distinct nt.tag as tag' .
       ' from grades g, group_inbox gr, notice_tag nt, notice n, profile p' .
       ' where g.noticeid = gr.notice_id' .
       ' and g.grade LIKE \'' . $grade . '\'' .
       ' and gr.group_id = ' . $idGroup .
       ' and gr.notice_id = nt.notice_id' .
       ' and n.id = nt.notice_id' .
       ' and n.profile_id = p.id' .
       ' and p.nickname LIKE \'' . $userid . '\'';
```

Para el *combobox* de usuarios, se ha creado la función *getUsersinGroupWithHashtagAndGrade*.

```
$qry = 'select p.nickname as nick' .
       ' from grades g, group_inbox gr, notice_tag nt, notice n, profile p' .
       ' where g.noticeid = gr.notice_id' .
       ' and g.grade LIKE \'' . $grade . '\'' .
       ' and gr.group_id = ' . $idGroup .
       ' and gr.notice_id = nt.notice_id' .
       ' and nt.tag LIKE \'' . $hashtag . '\'' .
       ' and n.id = nt.notice_id' .
       ' and n.profile_id = p.id' .
                   ' group by p.nickname';
```

Para el *combobox* de puntuaciones, se ha creado la función *getGradesinGroupWithTagAndUser*.

```
$qry = 'select distinct g.grade as grade' .
         ' from grades g, group_inbox gr, notice_tag nt, notice n, profile p' .
         ' where g.noticeid = gr.notice_id' .
         ' and gr.group_id = ' . $idGroup .
         ' and gr.notice_id = nt.notice_id' .
         ' and nt.tag LIKE \'' . $tag . '\'' .
         ' and n.id = nt.notice_id' .
         ' and n.profile_id = p.id' .
         ' and p.nickname LIKE \'' . $userid . '\'' .
         ' order by grade asc';
```

Un detalle muy importante en todas las consultas, es que como la información de un *combobox* va a depender de los otros dos, es importante que la consulta reciba el valor de los otros dos elementos de tipo *select*.

En el caso de que el valor de un *combobox* sea "Todos", o bien al rellenarlos por primera vez en el formulario, lo que se ha hecho es reemplazar esos dos campos adicionales con el carácter '%', ya que en SQL, esto se corresponde con cualquier cosa.

```
$tags = NotesPDF::getTagsOfUserWithGradeInGroup($this->idGroup,'%','%');
$nicks = NotesPDF::getUsersinGroupWithHashtagAndGrade($this->idGroup,'%','%');
$grades = NotesPDF::getGradesinGroupWithTagAndUser($this->idGroup,'%','%');
```

Para rellenar el elemento *select* con los datos obtenidos por las consultas, es necesario realizar un bucle *for*, en el que a medida que se van recorriendo los resultados, se vayan insertando en el *combobox*.

```
$this->out->element('option', array('value' => 'Todos'), 'Todos');
 for ($i = 0; $i < count($tags); $i++) {
            $this->out->element('option',   array('value'   =>   $tags[$i]),
$tags[$i]);
}
```

Una vez creado la acción y el formulario por completo, el resultado es el siguiente:

Actualización automática de opciones en los combobox

Ficheros modificados: Notescustomizeform.php, notes.js, updateBoxGrade.php, updateBoxTags.php, updateBoxUser.php

Uno de los grandes problemas de los apuntes manuales, es la necesidad de conseguir actualizar cada vez que se modifica el valor de un parámetro de personalización, todos los valores de los otros dos parámetros (combobox).

Para poder crear apuntes manuales sin que estos acaben en errores, es necesario actualizar en cada modificación de una de las cajas de selección las otras dos. Esto es, porque en un primer momento estos *combobox* se rellenan con toda la información disponible del grupo.

Es decir, se rellenan con todos los usuarios, todas las etiquetas y todas las puntuaciones existentes. Sin embargo, cuando un usuario seleccione en cualquiera de ellos algo, los valores de las otras dos cajas de selección deben variar.

Por ejemplo, al seleccionar en la caja de *Hashtag* una etiqueta determinada, es posible que ya no todos los usuarios tengan un mensaje con ese *hashtag*, o que ya no haya mensajes con todas las puntuaciones. Por ello, ha sido uno de los procesos más complejos realizados tanto en *BoloTweet*, como por supuesto en este *Plugin*.

Como los valores de las cajas de selección deben variar en tiempo de ejecución, y si es posible, sin recargar la página, se ha tenido que estudiar los detalles y el funcionamiento básico de AJAX para intentar utilizarlo.

Finalmente, la implementación ha sido la siguiente:

Como para poder ejecutar AJAX es necesario hacerlo a través de JavaScript, lo primero que se ha hecho es añadir un evento en cada combobox que permita ejecutar un script cada vez que se modifica un valor.

Por ello, se ha escogido el evento *onchange*, y se le ha añadido a cada *combobox* de la siguiente manera:

```
$this->out->elementStart('select', array('name' => 'combo-tag', 'id' => 'notes-combo-hashtag', 'class' => 'notes-combo-manual', 'onchange' => 'changeComboTag(' . $this->idGroup . ');'));
```

Aunque a primera vista el *groupid* parece que no es necesario, para poder posteriormente recargar las cajas de selección con los nuevos valores sí que se necesita, por lo que hay que pasárselo a las funciones de JavaScript.

En los otros dos elementos *select* el evento se ha incluido de la misma manera, únicamente modificando la función por la que corresponda, ya que cada uno tiene una función JavaScript propia.

Se han creado por tanto en el fichero *notes.js*, las funciones:

- changeComboTag
- changeComboUser
- changeComboGrade

La idea es la siguiente, en un primer lugar, se van a desarrollar unos scripts PHP que permitan obtener los valores correctos en cada *combobox* realizando consultas a la base de datos.

Este script va a ser llamado desde JavaScript utilizando la función *Load* de *JQuery*. Cada función JavaScript actualizará los otros 2 *combobox* restantes, actualizando su contenido cada vez que se modifique algún valor en cualquiera de ellos.

Por tanto, las funciones JavaScript creadas tienen todas el siguiente aspecto, variando únicamente en cada caso, los combobox que se recargan.

```
var tag = $('#notes-combo-hashtag').find(":selected").text();
var userid = $('#notes-combo-user').find(":selected").text();
var grade = $('#notes-combo-grade').find(":selected").text();

$('#notes-combo-user').load("../../local/plugins/NotesPDF/scripts/updateBoxUser.php", {groupid: groupid, grade: grade, tag: tag}, function() {

if (userid !== 'Todos') {
$("#notes-combo-user option[value=" + userid + "]").attr("selected", "selected");
        }
    });
    $('#notes-combo-hashtag').load("../../local/plugins/NotesPDF/scripts/updateBoxTags.php", {groupid: groupid, userid: userid, grade: grade}, function() {

if (tag !== 'Todos') {
```

```
$("#notes-combo-hashtag option[value=" + tag + "]").attr("selected", "selected");
}
    });
```

Como se puede ver, el primer paso en las tres funciones es obtener los valores actuales de las cajas de selección, en este caso utilizando la función find de JQuery junto con el parámetro ***:selected***.

Una vez obtenidos los valores, el siguiente paso es actualizar los otros dos comboBox restantes, dependiendo de la función que corresponda.

En este caso la función actualiza el combo de hashtag y el de user, por lo que esta función JavaScript corresponde al combo de puntuaciones.

Para actualizar el combo, lo que se ha hecho es lo siguiente. En un primer lugar seleccionamos el objeto select que corresponda a través de su ID, y utilizando los selectores de JQuery. Una vez seleccionado, hay que utilizar la función load de JQuery, que es una función abreviada para hacer una llamada Ajax, y que lo que permite es reemplazar el contenido del elemento seleccionado.

En este caso, el script deberá devolver en HTML las nuevas opciones del comboBox, y éstas se reemplazarán automáticamente en el elemento select.

Por otro lado, como se puede ver, en la llamada AJAX se realizan dos cosas más, la primera es que se pasan a través de la llamada los parámetros que va a necesitar el script, en este caso el groupid, que se pasa en todas las llamadas, y el tag y el grade en este caso.

La segunda cosa que se realiza es concatenar una función a la llamada, de manera que se ejecutará al finalizar la ejecución de la misma.

Como esta llamada AJAX se va a ejecutar en segundo plano, cuando el usuario modifique un comboBox, éste no verá nada raro en la página, pero internamente los valores de los otros comboBox se habrán actualizado.

Sin embargo, al actualizar mediante AJAX el elemento select, siempre se quedaba seleccionada la opción “Todos”, perdiendo el usuario la que tuviera.

Para solucionar esto, y ya que al comienzo del script se han obtenido los valores de cada comboBox, lo que se ha hecho es que utilizando esta función que se ejecutará al finalizar la llamada AJAX, seleccionar dentro de las opciones de cada comboBox el elemento que estaba antes seleccionado, y se marca como seleccionado.

Este proceso no es trivial, pues se ha tenido que utilizar un selector de JQuery, complejo,

```
$("#notes-combo-hashtag option[value=" + tag + "]")
```

Y lo que se ha hecho es añadirle el atributo “selected”.

En este caso hubo un problema, y es que al crear los comboBox en el formulario notescustomizeform, en un primer momento no se añadía el campo value a cada opción, únicamente se rellenaba el campo a mostrar.

Sin embargo, este selector funciona únicamente a través del campo value, por lo que se tuvo que modificar para añadirse como value el nombre que se muestra, y poder así posteriormente desde JQuery seleccionar cualquier opción.

```
$this->out->element('option', array('value' => $tags[$i]), $tags[$i]);
```

Una vez vista la parte JavaScript, queda la parte de los scripts.

Cuando se realiza la llamada AJAX, como destino están una serie de scripts que se encargan de obtener los nuevos valores de un campo comboBox, y de rellenarlo correctamente.

El código de los tres scripts es similar a este:

```
define('STATUSNET', true);
define('LACONICA', true); // compatibility
define('INSTALLDIR', realpath(dirname(__FILE__) . '/../../../..'));

require_once INSTALLDIR . '/lib/common.php';
require_once INSTALLDIR . '/local/plugins/NotesPDF/classes/NotesPDF.php';

$tag = ($_POST['tag'] == 'Todos') ? '%' : $_POST['tag'];
$userid = ($_POST['userid'] == 'Todos') ? '%' : $_POST['userid'];
$groupid = $_POST['groupid'];


$grades = NotesPDF::getGradesinGroupWithTagAndUser($groupid, $userid, $tag);

echo '<option value="Todos">Todos</option>';

for ($i = 0; $i < count($grades); $i++) {
    echo '<option value="' . $grades[$i] . '">' . $grades[$i] . '</option>';
}
```

En un primer lugar es totalmente imprescindible colocar la parte de los define, ya que de lo contrario se abortará la ejecución. Esto es debido a que como la ejecución del script es en segundo plano, no tiene las variables de ejecución almacenadas, de manera que si en este script no se definen una serie de variables que se comprueban al ejecutar la acción principal de StatusNet, se cancelará el script.

El siguiente paso es el *include* de los archivos que se vayan a utilizar en el script. Esto es muy importante, ya que como en este caso no hay nada en memoria, ni archivos cargados, ni funciones

autoLoad, ni nada por el estilo, ya que es un script independiente, todo aquello que no se referencie y se utilice dará un error.

El siguiente paso es obtener los valores pasados como parámetros a través de la llamada AJAX. En este caso, se puede ver como se utiliza el operador ternario para definir su valor. Esto es, porque al igual que se explicó en la tarea anterior, cuando la opción seleccionada por el usuario es "Todos", se debe sustituir por '%' antes de realizar la consulta SQL, ya que ese carácter se corresponde con cualquier cosa, y por tanto hará la función de "Todos".

El siguiente paso es la llamada a la función que corresponda, pasando como parámetros los obtenidos anteriormente. Estas funciones y sus consultas están explicadas en la tarea anterior.

Solamente mencionar que como un comboBox depende de los otros dos, es por ello por lo que es necesario pasarle a la consulta los valores de los otros dos, además del groupid, para obtener los valores actuales correctos.

Una vez obtenidos, es necesario generar un HTML que se corresponda con el interior del elemento select que se está recargando desde AJAX.

En este caso, se crea primero la opción "Todos", y posteriormente se rellena de las opciones obtenidas por la función anterior. Es importante aquí también incluir el campo value en cada opción, para poder luego desde JavaScript seleccionar un campo concreto y seleccionarlo.

Los 3 scripts son similares, variando los parámetros que le llegan, y la función a la que llaman, pero el funcionamiento es siempre el mismo.

Con todo esto, es posible actualizar automáticamente los valores de los elementos *select* a través de AJAX, permitiendo así generar apuntes completamente personalizados sin producir ningún riesgo, ya que si se pasa a la siguiente acción, se conoce que los campos seleccionados existen, y no van a producirse apuntes vacíos.

Crear acción que genere los ficheros PDF

Ficheros modificados: NotesPDFPlugin.php, NotesgenerateAction.php, NotesPDF.php, GenerarPDF.php

La última parte de este Plugin es crear la acción final, que reciba el formulario enviado por la acción NotescustomizeAction, y que genere a partir de los parámetros recibidos un fichero PDF con apuntes.

Para ello, el primer paso es crear el fichero correspondiente *NotesgenerateAction.php*, e incluirlo como siempre en su respectiva función onAutoLoad de NotesPDFPlugin.

Como no va a haber una acción posterior, ya que esta es la que realiza la generación final, no va a ser necesario crear ningún formulario.

Lo primero que se hace en la función *handle* de la acción es obtener el *groupid*, pasado como parámetro, y hacer la distinción entre apuntes automáticos o personalizados.

Tal y como se explicó en la tarea *Crear acción para seleccionar tipo de apuntes*, como el formulario de selección de apuntes envía un *submit* diferente dependiendo del tipo de apuntes, es necesario en este caso realizar la distinción, para poder aplicar una u otra acción.

En el caso de que el *submit* coincida con los apuntes automáticos, se realiza lo siguiente:

```
if ($this->trimmed('submit-auto') != null) {
            $groupids = NotesPDF::getNoticeIDsInAGroupModeAuto($idGroup);
            $notices = Notice::multiGet('id', $groupids)->fetchAll();
GenerarPDF::content($idGroup, $notices, 'Automáticos');
}
```

Como en los apuntes automáticos se seleccionan por defecto las ideas con una puntuación de 2, o 3 puntos, es posible crear una consulta que devuelva tales mensajes directamente.

La función que realiza esta consulta es getNoticeIDsInAGroupModeAuto, y está implementada de la siguiente forma:

```
$qry = 'select distinct g.noticeid as noticeID' .
                ' from grades g, group_inbox gr' .
                ' where g.noticeid = gr.notice_id' .
                ' and gr.group_id = ' . $idGroup .
                ' and (g.grade = 2 or g.grade = 3)' .
                ' order by g.noticeid';
```

Una vez realizada la llamada a esta función, y obtenido el array con los ids de las noticias seleccionadas, se ha utilizado la función MultiGet de Notice, para obtener los objetos *Notice* correspondientes a los ids seleccionados.

Por último, una vez obtenido el array de noticias, tan sólo queda llamar a una función que se encargue de generar el archivo PDF, *content*.

En el caso de que los apuntes sean personalizados, el código es similar, variando únicamente la consulta y la obtención de parámetros:

```
else if (($this->trimmed('submit-custom') != null)) {


$tag=$this->trimmed('combo-tag')=='Todos'?'%':$this->trimmed('combo-tag');
$nick=$this->trimmed('combo-user')=='Todos'?'%':$this->trimmed('combo-user');
```

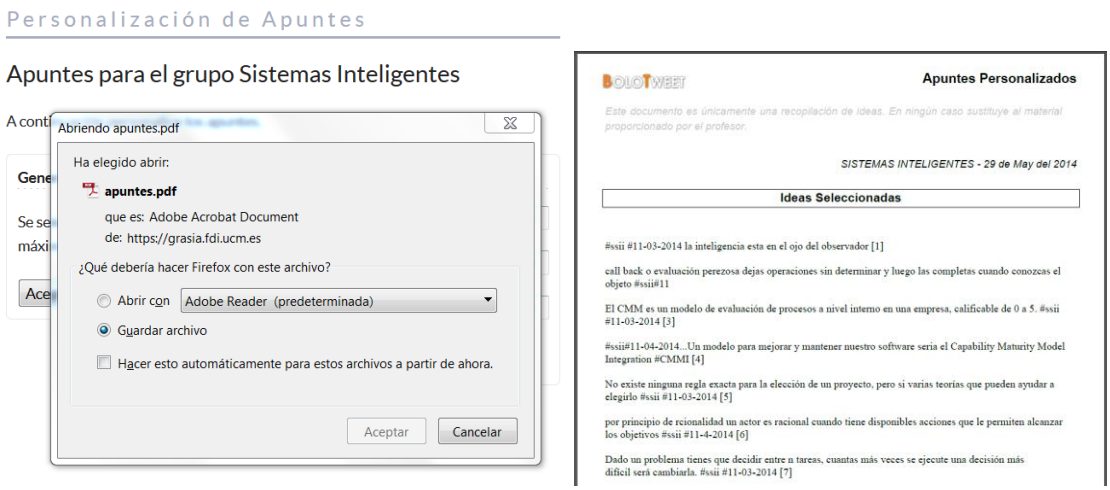
```
$grade=$this->trimmed('combo-grade')=='Todos'?'%':$this->trimmed('combo-grade');


$noticeIds = NotesPDF::getNoticesInModeCustom(array('idGroup' => $idGroup, 'tag' =>
$tag, 'nick' => $nick, 'grade' => $grade));
$notices = Notice::multiGet('id', $noticeIds)->fetchAll();
GenerarPDF::content($idGroup, $notices, 'Personalizados');


}
```

En este caso se obtienen los valores de los *comboBox*, sustituyéndolos por '%', en el caso de que la opción seleccionada sea "Todos", para poder realizar la consulta SQL correctamente.

```
$qry = 'select distinct g.noticeid as noticeid' .
                    ' from grades g, group_inbox gr, notice_tag nt, notice n,
profile p' .
                    ' where g.noticeid = gr.notice_id' .
                    ' and g.grade LIKE \'' . $grade . '\'' .
                    ' and gr.group_id = ' . $idGroup .
                    ' and gr.notice_id = nt.notice_id' .
                    ' and nt.tag LIKE \'' . $tag . '\'' .
                    ' and n.id = nt.notice_id' .
                    ' and n.profile_id = p.id' .
                    ' and p.nickname LIKE \'' . $nick . '\'';
```

A continuación, una vez obtenidos los IDs de las noticias que correspondan, se utiliza de nuevo el método MultiGet para obtener los objetos *Notice*, y finalmente se llama a la función correspondiente para generar el fichero PDF, en este caso, *content*.

El resultado final en ambos casos es el siguiente:



Crear clase que se encargue de generar el archivo PDF

Ficheros modificados: GenerarPDF.php

Para que a partir de la última acción se genere el fichero PDF, es necesario crear una clase que se encargue de crearlo.

Esta clase es *GenerarPDF*, y es la que se encarga de todo lo relacionado con la librería FPDF y con la generación del PDF.

Lo primero que se va a hacer es explicar brevemente el comportamiento de la librería FPDF.

Esta librería, a la que se accede a través del archivo principal *fpdf.php*, dispone de una serie de funciones predefinidas para realizar acciones sobre los ficheros PDF.

Las funciones más importantes que se han utilizado durante todo el proceso son las siguientes:

- AddPage -> Añade una nueva hoja al archivo PDF.
- Image -> Añade imágenes al archivo PDF
- SetFont -> Fija una determinada fuente para todo el texto posterior.
- SetTextColor -> Fija el color de la fuente para todo el texto posterior.
- Cell -> Añade una sola línea de texto al fichero PDF.
- MultiCell -> Añade más de una línea de texto al fichero PDF.
- Write -> Añade texto y lo ajusta en el fichero PDF, independientemente de su longitud.
- Ln -> Añade saltos de línea.
- SetY -> Fija la posición de un elemento en la página

Por otro lado, también dispone de una serie de métodos que se deben sobrescribir para proporcionar una mejor apariencia al fichero PDF final, como el método *Header*, para añadir una cabecera a todas las páginas del PDF, o el método *Footer*, para incluir un pie de página.

Para poder sobrescribir sus métodos, es necesario que la clase extienda de la librería.

class GenerarPDF extends FPDF { … }

Una vez entendido el funcionamiento básico de la clase, se van a explicar qué métodos se han implementado.

El primer método es el constructor, y se ha sobrescrito para poder añadir un atributo interno “modo”, que permita posteriormente en el método “*Header*” distinguir entre ambos tipos de apuntes.

```
function __construct($modo = '') {
        parent::FPDF();
$this->modo = $modo;
    }
```

El siguiente método que se ha implementado es *Header*. Con esta función se podrá crear una cabecera en cada nueva página del fichero PDF.

```
function Header() {
	$this->Image(Theme::path('logo.png'), 10, 8, 33);
	// Arial bold 15
	$this->SetFont('Arial', 'B', 15);
	// Movernos a la derecha
	$this->Cell(80);
	// Título
	$this->Cell(0, 0, 'Apuntes ' . $this->modo, 0, 1, 'R');
	$this->Ln(10);
	$this->SetFont('Arial', 'I', 12);
	$this->SetTextColor(199, 199, 199);
	$this->MultiCell(0, 6, 'Este documento es únicamente una recopilación de
	ideas. En ningún caso sustituye al material proporcionado por el profesor.',
	0);
	$this->Ln(5);
    }
```

Como se puede ver, se ha añadido incluido el logo de BoloTweet, y una serie de texto informativo. Además, se puede ver cómo se ha utilizado el atributo "modo" para generar el *header* personalizado dependiendo del tipo de apuntes.

Por otro lado, la función Footer también se ha sobrescrito, y lo único que se ha indicado es el número de página, utilizando la función interna de FPDF, PageNo.

```
	function Footer() {
		$this->SetY(-15);
		$this->SetFont('Arial', 'I', 8);
		$this->Cell(0, 10, 'Page ' . $this->PageNo(), 0, 0, 'C');
	}
```

Una vez creados estos métodos, se ha añadido una nueva función nueva para añadir una portada.

Esta función es común a ambos tipos de apuntes, y lo único que hace es añadir el nombre del grupo para el que se están generando los apuntes, la fecha actual, y un título informativo.

```
function Portada($idGroup) {

	$groupName = NotesPDF::getGroupByID($idGroup)->getBestName();

	$this->AddPage();
	$this->Ln(7);
	$this->SetFont('Arial', 'I', 12);
	setlocale(LC_ALL, "es_ES");
	$this->Cell(0, 0, strtoupper($groupName) . ' - ' . strftime("%d de %B del
%Y"), 0, 0, 'R');
```

```
$this->Ln(10);
        $this->SetFont('Arial', 'B', 14);
        $this->Cell(0, 7, 'Ideas Seleccionadas', 1, 0, 'C');
        $this->Ln(10);
}
```

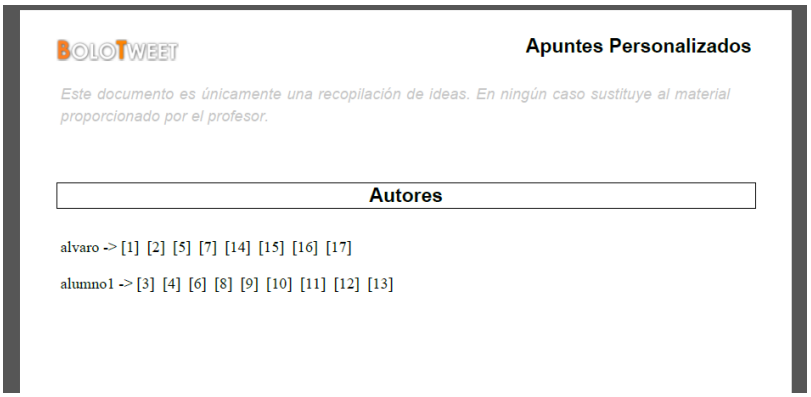
Por último, en cuanto a funciones que complementan los apuntes, se ha añadido la función Fuentes, cuyo objetivo es añadir una última página con información sobre los autores de cada idea mostrada en los apuntes. De esta manera, se podrá observar con facilidad quién es el autor de una noticia en concreto.

```
function Fuentes($authorAndNotice, $authors) {

        $this->addPage();
        $this->SetFont('Arial', 'B', 15);

        $this->Ln(10);

        $this->Cell(0, 7, 'Autores', 1, 0, 'C');
        $this->Ln(5);
        $this->SetFont('Times', '', 12);

        foreach ($authors as $author) {

            $noticeIds = array_keys($authorAndNotice, $author);

            $linea = $author . " ->";

            foreach ($noticeIds as $id) {

$linea .= " " . "[" . $id . "] ";
            }

            $this->Ln(10);
            $this->Write(5, $linea);
}
    }
```

Esta función recibe un *array* con autores, y por otro lado un array asociativo con autor más número de idea.

Lo que se hace es recorrer el *array* de autores, y para cada uno de ellos, se obtiene a partir del *array* asociativo todos los números de sus ideas, utilizando la función de PHP *array_keys*.

De esta manera, es posible realizar un listado en el que se muestre por cada autor con su conjunto de ideas.



Una vez explicadas todas estas funciones auxiliares, tan sólo queda explicar la función que se encarga de utilizarlas todas para generar el fichero PDF, la función *content*.

Esta función recibe el *groupid*, necesario para poder obtener el nombre del grupo al crear la portada. El *array* con las noticias, para poder incluirlas en los apuntes, y por último el *modo*, para generar la cabecera personalizada.

Su funcionamiento, una vez creadas el resto de funciones, es relativamente sencillo.

En primer lugar se crea el archivo PDF con el constructor mostrado anteriormente, y se le pasa el parámetro *modo* para que lo almacene en el atributo de clase.

Posteriormente se llama a la función portada, y a continuación solamente queda empezar a insertar las ideas seleccionadas.

Para ello, y pensando en la función Fuentes en la que se debe mostrar el listado de autores junto con sus ideas, es necesario crear 2 *arrays*, en los que ir almacenando por un lado los autores, y por otro lado los autores junto con su nota.

Tras escribir todas las noticias, se llama a la función Fuentes con los *arrays* creados anteriormente, y en el caso del array de autores se utiliza la función de PHP *array_unique*, para dejar solamente una vez cada nombre de alumno.

Finalmente, se llama a la función *Output* de la librería FPDF para generar el archivo de salida. En este caso se ha escrito un nombre predefinido, y se ha seleccionado la opción de descarga ‘D’, para que se genere una ventana de descarga del fichero en el navegador del usuario.

El código utilizado en esta función es el siguiente:

```
static function content($idGroup, $notices, $modo) {

        $pdf = new GenerarPDF($modo);
        $pdf->Portada($idGroup);
        $pdf->SetFont('Times', '', 12);
```

```
        $i = 1;
        $authorAndNotice = array();
        $authors = array();

        foreach ($notices as $notice) {
            $pdf->Ln(10);
            $filterContent = $pdf->filtrarContenido($notice->content);
            $filterContent = $filterContent . " [" . $i . "]";
            $pdf->Write(5, $filterContent);

            $authorAndNotice[$i] = $notice->getProfile()->getBestName();
            $authors[] = $notice->getProfile()->getBestName();
            $i = $i + 1;
        }


        $pdf->Fuentes($authorAndNotice, array_unique($authors));
$pdf->Output('apuntes', 'D');
    }
```

Un detalle importante, es que a la hora de escribir las noticias en el PDF, como se puede ver se han procesado con una función *filtrarContenido*. Esta función se ha creado para eliminar de las noticias las menciones a grupos, de manera que si en un tweet existía el siguiente contenido:

!ssii Esto es una prueba de filtrado #pruebaFiltrado.

Únicamente se selecciona el contenido relevante del mensaje:

Esto es una prueba de filtrado #pruebaFiltrado.

Para conseguir esto, se ha decidido utilizar la función de PHP *preg_replace*. Esta función reemplaza una expresión regular por lo que los caracteres que se le indique.

De esta manera, si se consigue desarrollar una expresión regular que seleccione cualquier palabra que empiece por '!', y sustituirla por '', se podrán eliminar las menciones a grupos del contenido del mensaje.

Para probar de manera rápida las expresiones regulares, se ha utilizado el siguiente servicio online.

http://gskinner.com/RegExr/

Finalmente, la expresión regular que permite seleccionar cualquier palabra que empiece por el carácter '!' independientemente de donde esté situada en el texto, es la siguiente:

/(?:^|\s)!\w{1,64}/

De esta manera, el contenido de la función filtrar contenido es el siguiente:

```
function filtrarContenido($content) {
        return ltrim(preg_replace('/(?:^|\s)!\w{1,64}/', '', $content));
}
```

Como se puede ver, también se aplica la función *ltrim*, en este caso lo que se consigue con esto es borrar cualquier espacio en blanco que haya podido quedar al comienzo del mensaje.

Los ficheros PDF generados no son reconocidos como tal en Windows

Ficheros modificados: GenerarPDF.php, fpdf.php

Para solucionar esto, se han tenido que realizar dos sencillos pasos.
El primero ha sido modificar el nombre del fichero PDF generado en la clase *GenerarPDF.php*, de manera que ahora además del nombre se ha añadido la extensión, y en Windows ya es posible abrirlos.

```
$pdf->Output('apuntes.pdf', 'D');
```

Por otro lado, lo que se ha hecho es modificar algunas de las cabeceras internas de la librería FPDF que se utilizan al generar el fichero de descarga.

Por defecto, en la clase fpdf.php, en la función Output, se utilizaba la siguiente cabecera para reconocer el fichero de descarga:

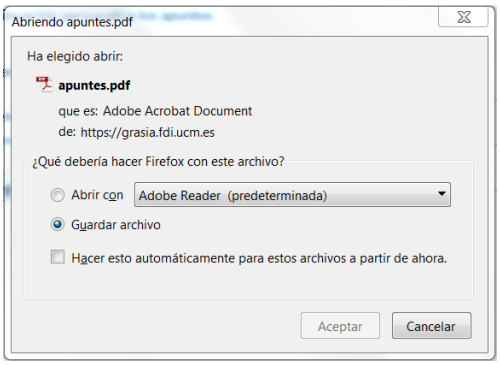
```
header('Content-Type: application/x-download');
```

Sin embargo, esto provocaba que el fichero generado no fuera reconocido como un archivo PDF por el navegador, no pudiendo por ejemplo, abrirlo a través del mismo.

Para corregir esto, se ha tenido que modificar esa cabecera, sustituyéndola por:

```
header('Content-Type: application/pdf');
```

De esta manera, ahora sí se reconoce como documento PDF, y podría ser abierto a través del visor PDF del navegador:



Errores en palabras con caracteres especiales

Ficheros modificados: fpdf.php

En los apuntes generados a través de la clase FPDF, si alguna palabra contiene caracteres especiales, se muestran mal codificadas.

Para corregirlo, se ha utilizado la función de PHP iconv, que permite convertir una cadena dada de una codificación a otra.

Una de las opciones que se barajaron era utilizar esta función en cada string, pero era demasiado incómodo. Se optó también por crear una función codificar, que realizara la conversión, pero no era muy cómodo de todas maneras tener que llamar a esta función por cada cadena.

De esta manera, finalmente se ha decidido modificar la librería FPDF, de manera que internamente se codifique cualquier cadena que le llegue.

Para hacer esto, y tras estudiar el funcionamiento de la librería, cualquier función de escritura en el fichero pasa por la función Cell, por lo que es ahí donde se ha añadido la conversión de codificación, y de esta manera el problema está resuelto.

```
$txt = iconv("UTF-8", "ISO-8859-1", $txt);
```

Al pulsar en Apuntes, no se queda marcada la opción

Ficheros modificados: NotesPDFPlugin.php

El error estaba producido por la manera de generar el *menu_item*.

La solución es la siguiente:

- En el método *onStartDefaultLocalNav*, hay que obtener el nombre de la acción que se está ejecutando:

```
$actionName = $action->trimmed('action');
```

- Una vez hecho esto, al crear el menu_item, en el campo correspondiente a *selected*, hay que añadir lo siguiente:

```
($actionName === 'notesgroups' || $actionName === 'notescustomize')
```

- De esta manera, si el nombre de la acción coincide con notesgroups, o notescustomize, que son los nombres de las acciones que se utilizan en este plugin, ese campo se sustituirá por *true*, y por tanto el enlace aparecerá seleccionado.

Cambiar enlace Apuntes para utilizar el nuevo hook onStartToolsLocalNav

Ficheros modificados: NotesPDFPlugin.php

Con la aparición del plugin Task, ha sido necesario modificar la manera de crear el enlace de apuntes. Ahora es necesario utilizar el nuevo hook*onStartToolsLocalNav*, de manera que ahora todo el código de la tarea *Añadir un submenú en el lateral de la página, con un enlace a Apuntes*, ha sido sustituido por el siguiente:

```
function onStartToolsLocalNav($action) {
        $actionName = $action->trimmed('action');


        $user = common_current_user();
        if (!empty($user)) {
            $action->menuItem(common_local_url('notesgroups'),        _m('Apuntes'),
_m('Apuntes   en   PDF'),   ($actionName   ===   'notesgroups'   ||   $actionName   ===
'notescustomize'), 'nav_notespdf');
}
        return true;
    }
```

Como se puede ver, ya no es necesario definir el código, ni la lista, ni nada similar, ya que se hace en el *core* de *BoloTweet*.

SecureRegistration:

La idea de este plugin es proporcionar un mecanismo de registro seguro, con campos antiSpam, y con un *captcha*, que evite el registro de *bots* en el sistema.

Aprovechando que se ha reemplazado el formulario de registro predeterminado, se han dejado únicamente los campos que se han considerado oportunos, eliminando aquellos innecesarios.

NOTA: A pesar de estar de incluir el plugin en el sistema, se encuentra por defecto desactivado, ya que el registro se controla únicamente por el Administrador.

Crear la estructura interna del plugin

Ficheros modificados: SecureRegistrationPlugin.php, secureregistration.css

Como en el resto de Plugins, es necesario crear una estructura aislada propia, con las carpetas y archivos que se van a necesitar.

En este únicamente va a ser necesario el fichero principal del plugin, *SecureRegistrationPlugin.php*, y las carpetas *actions*, *css* y *lib*.

Proceso para sustituir el registro por defecto

Ficheros modificados: SecureRegistrationPlugin.php

En este caso, como lo que se quiere es reemplazar algo existente, y no incluir cosas nuevas, el mecanismo va a ser completamente diferente.

Para entender este funcionamiento, se ha utilizado como base el plugin EmailRegistration, que se encarga de reemplazar el formulario por defecto de registro únicamente por un correo electrónico.

De esta manera, entendiendo este Plugin, se podrá observar cómo consigue reemplazar el formulario por defecto.

Tras estudiarlo en detalle, la solución es la siguiente:

Lo primero que hay que utilizar es el evento *onArgsInitialize*, que nos va a permitir realizar acciones nada más terminar de inicializar el array con argumentos, es decir, antes de llamar a ninguna acción.

El siguiente paso será comprobar si en el array de argumentos existe un elemento de tipo 'action', y si el valor de ese elemento es 'register'. De esta manera, y ya que el parámetro que recibe este *hook* viene por referencia, podremos modificar el array de argumentos, y cambiar así el valor de ese elemento action, para redirigirlo a la nueva acción.

function onArgsInitialize(&$args){

```
if (array_key_exists('action', $args) && $args['action'] == 'register') {
$args['action'] = 'secureregister';}
return true;
}
```

Por otro lado, va a ser necesario utilizar el evento onLoginAction, con el que se debe indicar que esta acción es de tipo "login".

```
function onLoginAction($action, &$login)      {
      if ($action == 'secureregister') {
      $login = true;
      return false;       }
      return true;
      }
```

Con esto, se conseguirá "secuestrar" la acción *register*, para que se pueda enlazar con la nueva acción.

Buscar módulo captcha compatible y sencillo de utilizar

Ficheros modificados: <Ninguno>

Tras barajar diferentes posibilidad, finalmente se ha decidido utilizar el módulo *recaptcha*, proporcionado por Google, ya que es el más utilizado, con más soporte, y sencillo de utilizar.

http://www.google.com/recaptcha

El siguiente paso ha sido buscar información sobre el desarrollo en PHP utilizando este módulo. En la sección de desarrolladores de Google, existe un *Quick Start* con la información necesaria para comenzar a utilizarlo.

https://developers.google.com/recaptcha/docs/php?hl=es&csw=1

Tal y como indican en la guía de inicio rápido, lo primero necesario es la librería *recaptchalib.php*. Por tanto, se ha descargado y se almacenado en la carpeta *lib* del *plugin*.

Por otro lado, como la librería recaptchalib no es más que un wrapper que trabaja y se comunica con la API del servicio reCAPTCHA, ha sido necesario adquirir la API Key para poder utilizar este servicio. Se han obtenido por tanto un par de claves (privada y pública) para el dominio de la página. En este caso, se ha escogido como dominio el servidor de GRASIA (grasia.fdi.ucm.es), de manera que esta clave sea válida para cualquier subdominio.

```
Public Key:     6LfbNe0SAAAAAMcGjQh8HLN75YZ4Agckygg0bLcV
Use this in the JavaScript code that is served to your users

Private Key:     ------- This key is a secret ----------
Use this when communicating between your server and our server. Be sure to keep it
a secret.
```

Una vez conseguido conseguida la clave y la librería, tan sólo falta utilizarla en la acción. Según la guía de inicio rápido, el *captcha* se debe añadir en el lado del cliente a través de JavaScript, mientras que la comprobación del mismo, debe realizarse en el servidor al enviar el formulario con el resto de datos.

Crear acción que se encargue del registro seguro

Ficheros modificados: SecureRegistrationPlugin.php, SecureregisterAction.php, secureregister.css

Cuando se “secuestra” la acción register por defecto, es para enviarla a la nueva acción que se encargue de tratar y gestionar el nuevo registro.

Es por ello por lo que es necesario crear el fichero en el que se va a implementar el comportamiento de la acción, *SecureregisterAction.php*.

Esta acción se va a encargar de mostrar el formulario con los nuevos campos de registro, de añadir y controlar el módulo *reCAPTCHA*, y por último de realizar todo el proceso de registro.

Es importante mencionar que esta acción se ha basado completamente en la propia acción de registro de StatusNet, *register.php*, y se han ido modificando y añadiendo las cosas que se han considerado oportunas.

En la función *prepare* de la acción, tal y como realiza la acción register.php, se han controlado todo tipo de circunstancias.

Como *StatusNet* soporta registros abiertos, cerrados, por invitaciones… en una acción que controle el registro en el sistema, es necesario tratar todos estos puntos.

Por otro lado, en la función *handle*, es donde se ha implementado todo el control del nuevo registro seguro, por lo que se va a tratar de explicar por pasos.

```
function handle($args) {
        parent::handle($args);
        if (common_config('site', 'closed')) {
            // TRANS: Client error displayed when trying to register to a closed
site.
            $this->clientError(_('Registration not allowed.'));
        } else if (common_logged_in()) {
            // TRANS: Client error displayed when trying to register while already
logged in.
            $this->clientError(_('Already logged in.'));
        } else if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            $this->tryRegister();
        } else {
            $this->showForm();
}
    }
```

El código es el mismo que en la clase register, y lo único que se hace es realizar comprobaciones para saber qué pasos realizar.

Si los registros están cerrados, se mostrarán un error indicándolo.

Si el usuario ya está logueado en el sistema, no se puede registrar un nuevo usuario.

Si la llamada a esta acción se ha producido a través de POST, entonces quiere decir que ya se ha rellenado el formulario, y se debe por tanto intentar registrar el nuevo usuario.

Por último, si no son ninguna de las opciones anteriores, lo que hay que hacer es mostrar el formulario de registro.

En este caso las que se han modificado han sido, showForm, para personalizar el formulario, y tryRegister, para añadir todo el tratamiento del *captcha*.

En la función showForm, lo único que se hace es llamar a la función showPage, y es en los métodos sobrescritos de la clase donde se muestra el contenido de la página, en este caso, en el método showFormContent.

En esta función se ha creado el formulario de la misma forma que en la acción register.

Un detalle importante, es que la acción vinculada a este formulario sigue siendo "*register*", ya que si se coloca la nueva acción "*secureregister*", se producirá un error. Esto es, porque para poder ejecutar esta nueva acción, lo que se ha hecho es secuestrar la acción *register*, y redirigirla a este nuevo fichero.

Sin embargo, no se ha creado en ningún lado la función onRouterInitialized, en donde se vincule una acción con una url, por lo que la acción secureregister no tiene ningún fichero asociado. Por ello, la única manera de llegar a este fichero es secuestrando esa llamada, y redirigiéndola a la nueva acción, obteniendo el resultado esperado.

Los campos que se han decidido mantener son los siguientes:

- Nickname
- Contraseña
- Email
- Nombre Completo

Se han quitado los campos Localización, Biografía y *Homepage*, pudiendo rellenarse posteriormente si se desea a través de su perfil.

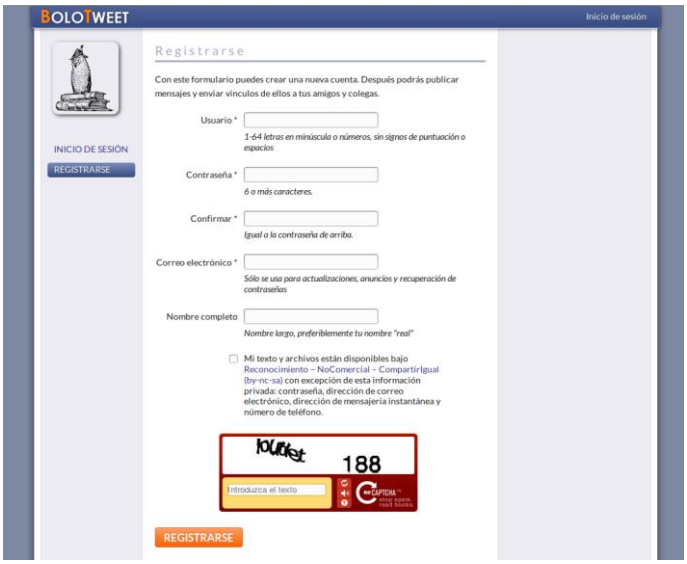
Se ha mantenido el checkBox para aceptar la licencia, y se ha añadido la parte en que se muestra el captcha. Además, se ha añadido un asterisco en los campos obligatorios, todos menos el nombre completo, para que el usuario lo sepa a priori, sin tener que enviar el formulario y observar los errores.

En cuanto a la inserción del *captcha*, según la guía de inicio rápido, el proceso consiste en incluir la librería en el fichero, definir la clave pública, y posteriormente llamar a la función *recaptcha_get_html* con un *echo*, que lo que hace es añadir un *inline script*.

Sin embargo, como en este caso los elementos en BoloTweet se incluyen a través de funciones, el comando *echo* no va a funcionar. Tras varias pruebas, la solución estaba en utilizar la función $this->raw, que lo que permite es incrustar en la página el código que se le pasa como parámetro tal cual, lo que emularía el comportamiento del *echo*.

```
$this->elementStart('li');
    $publickey = "6LfbNe0SAAAAAMcGjQh8HLN75YZ4Agckygg0bLcV";
    $this->raw(recaptcha_get_html($publickey));
$this->elementEnd('li');
```

Efectivamente, si se ejecuta la página, se ha modificado el formulario, y se ha añadido al final de la lista un nuevo elemento con el *captcha*. Mediante reglas CSS se ha colocado el formulario, y el resultado es el siguiente:



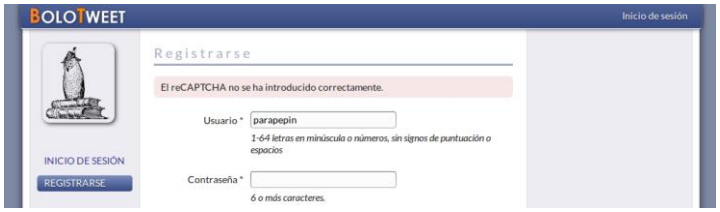
Una vez creada esta parte, el siguiente paso es el tratamiento de los datos del formulario, incluido el tratamiento del *captcha* a través de la función *trySave*.

Como si el reCAPTCHA está mal relleno no se debe continuar con el proceso de validación de datos, es preciso realizar la comprobación antes de nada. La comprobación se debe realizar en el lado del servidor utilizando la función *recaptcha_check_answer*.

Para poder validar el captcha introducido, esta función necesita conocer la clave privada. Posteriormente, y una vez comprobada la respuesta, deberá comprobarse si es o no válida, y dependiendo del resultado abortar el registro, o bien continuar con el tratamiento de los datos.

```
            $privatekey = "--------------------------------------";
            $resp    =    recaptcha_check_answer($privatekey,    $_SERVER["REMOTE_ADDR"],
            $_POST["recaptcha_challenge_field"], $_POST["recaptcha_response_field"]);
if (!$resp->is_valid) {
                $this->showForm("El captcha no se ha introducido correctamente."));
            } else { … }
```

Como se puede ver en el código, se han realizado las indicaciones de la ayuda del módulo reCAPTCHA. Primero se define la clave privada, y posteriormente se llama a la función*recaptcha_check_answer*. Dependiendo del resultado de ésta, se muestra un mensaje de error utilizando el mecanismo ya explicado en otras tareas de showForm junto con showPageNotice, o bien se prosigue con la validación de los datos.



Si el captcha se rellena correctamente, el siguiente paso es obtener todos los datos del formulario con la función trimmed, excepto en el campo password, que se permite que una contraseña tenga espacios, por lo que se usa la función arg.

```
$nickname = $this->trimmed('nickname');
                $email = $this->trimmed('email');
                $fullname = $this->trimmed('fullname');
                $password = $this->arg('password');
$confirm = $this->arg('confirm');
```

En cuanto a las comprobaciones de los datos, se han incluido únicamente las de los campos que se han mantenido:

- Nick con caracteres no válidos
- Campo de licencia aceptado
- Campo email vacío.
- Campo email con un buen formato.
- Nick ya existente
- Nick bloqueado
- Email existente
- Nombre completo demasiado largo
- Contraseña de menos de 6 caracteres

Una vez realizadas todas estas comprobaciones, se pasa a registrar el usuario utilizando la función *User::register*.

Si el registro ha sido satisfactorio, se mostrará un mensaje indicándolo, y si ha sucedido algún error, también se anunciará.

Creado nuevo sistema de protección contra SPAM

Ficheros modificados: SecureregisterAction.php, secureregister.css

Al cabo de unos pocos días tras insertar el módulo *reCAPTCHA*, una oleada de *bots* se lo consiguieron saltar registrándose de nuevo en el sistema y llenándolo de SPAM.

Por ello, ha sido necesario incluir otro sistema de protección adicional antiSPAM. En este caso, tras buscar alternativas y soluciones de todo tipo, se optó por utilizar una solución sencilla pero muy eficaz.

La idea es la siguiente:

Cuando se registran *bots*, lo que hacen es coger el código HTML del formulario, comenzar a rellenar, y finalmente pulsar el botón enviar. Sin embargo, estos ataques de registro están programados de manera que se rellenen todos los campos del formulario.

Aquí es donde reside la protección antiSPAM. Como los *bots* van a rellenar todos los campos del formulario, la idea es generar un elemento input adicional, en este caso correspondiente al teléfono, y ocultarlo mediante CSS.

De esta manera, un usuario real completará el resto de campos, y se registrará satisfactoriamente. Por el contrario, los *bots* cogerán el HTML, y comenzarán a completar todos los campos, incluido el oculto.

De esta manera, posteriormente lo que hay que hacer es realizar una comprobación al recibir el formulario enviado, de manera que si el campo correspondiente al teléfono está relleno, se aborte el registro.

Para implementar esto, ha sido tan sencillo como incluir en el formulario un nuevo elemento "trampa":

```
$this->elementStart('li', array('id' => 'phone'));
            $this->input('phoneLbl', _('Phone'), $this->trimmed('phone'));
$this->elementEnd('li');
```

Posteriormente, en la función *trySave*, que es la que recibe el formulario de registro, lo que se ha hecho es controlar este campo:

```
            if ($this->trimmed('phoneLbl') != "") {
                return;
            }
```

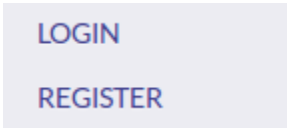
Con este método se consiguió estar alrededor de un mes sin un solo usuario SPAM registrado. No obstante, al cabo del mes algunos usuarios comenzaron a saltárselo, pues ya debían de haber aprendido qué campo era el que no debían rellenar.

Una posible solución a esto es crear el campo antiSPAM con un ID aleatorio, utilizando de manera que por ejemplo al ID fijo, cada vez que se ejecutara la acción se le concatenara un número aleatorio utilizando la función ***rand*** de PHP.

Sin embargo, como los registros deben estar cerrados por defecto, ya que el alta de usuarios es tema administrativo, esta última opción no se pudo llegar a probar. No obstante, y como es lógico, con los registros cerrados no ha vuelto a haber ningún ataque de SPAM.

No se queda seleccionado el enlace Register

Ficheros modificados: logingroupnav.php



Este problema se ha resuelto anteriormente, en este caso en la tarea *Al pulsar en Apuntes, no se queda marcada la opción.*

El problema es producido por lo mismo, pero se resuelve de diferente forma. En el caso de la tarea anteriormente citada, el *menu-item* se estaba implementado de manera incorrecta, y por tanto faltaba indicar en el campo selected cuándo se debía poner a *true*.

En este caso, el problema es que como el enlace de *register* se ha creado para la acción register, el campo selected utiliza el nombre de acción register, mientras que ahora al redirigirla a la nueva acción *secureregister*, esa comprobación nunca es cierta, y por tanto ese campo nunca está a *true*.

El archivo que se ha modificado es *lib/logingroupnav.php*. Cuando se genera el menu_item, solamente ha habido que añadir como posible nombre de acción *secureregister*, de manera que no es necesario eliminar el ya existente, y de esta manera es posible utilizar o no el plugin, que en ambos casos el enlace va a quedar seleccionado.

```
$this->action->menuItem(common_local_url('register'),
       _m('MENU','Register'),
       _('Sign up for a new account'),
        ($action_name === 'register' || $action_name === 'secureregister');
```

LogoPlugin:

La idea de este plugin es proporcionar un espacio para el nuevo logo de BoloTweet. Este logo se debe de poder visualizar de manera correcta en cualquier página de BoloTweet.

Crear la estructura del plugin

Ficheros modificados: LogoPlugin.php, logo.css

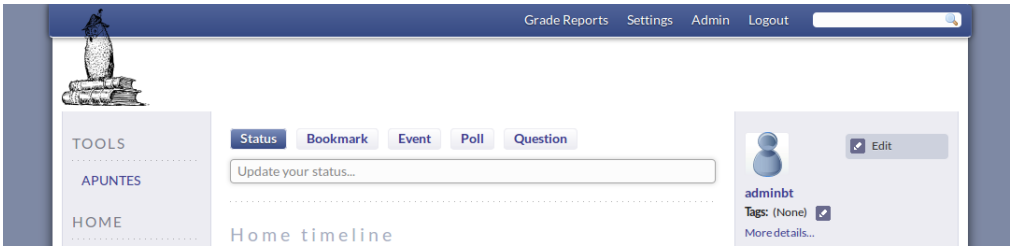
Como este plugin no va a necesitar ningún tipo de acción, ya que solamente va a utilizar un *hook* para insertar el logo, solamente ha hecho falta crear la carpeta css, para añadir las reglas que se consideren oportunas, y el archivo principal del plugin *LogoPlugin.php*.

Buscar un sitio apropiado para el logo

Ficheros modificados: <Ninguno>

Tras buscar un lugar en el que colocar el nuevo logo, la primera idea era situarlo en el sitio original del logo de StatusNet.

Sin embargo, este logo ocupa un espacio totalmente diferente al reservado para el logo original de BoloTweet, por lo que situarlo ahí provocaría que se descuadrara por completo la página.



Por ello, y tras barajar distintas opciones, se ha decidido utilizar el hook *StartDefaultLocalNav*, y colocar el logo justo encima del menú "Tools", de manera que no descuadre el contenido de la página, y sin embargo se muestre de forma correcta en todo momento.

Añadir el logo utilizando el hook StartDefaultLocalNav

Ficheros modificados: LogoPlugin.php

Una vez localizado el mejor sitio para situar el logo, el siguiente paso es crear en el archivo *LogoPlugin.php* la función que utilice tal evento, *onStartDefaultLocalNav*.

Como este evento permite acceder elementos dentro de la lista del *aside* izquierdo, es importante que el elemento imagen vaya dentro bloque <li>, que representará un nuevo elemento de la lista.

Además de mostrar la imagen, también se ha querido proporcionar el mismo comportamiento que el logo original, de manera que al pulsar en ella, se muestre la página de inicio.

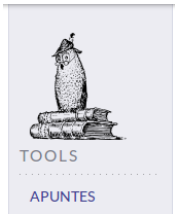
Para ello, lo que se ha hecho es intentar reconstruir la misma estructura que la utilizada en el logo original, es decir, dentro del elemento <li>, un elemento <address> de la clase vcard, posteriormente un elemento <a>, con las clases "url home bookmark", y por último el elemento <img> con las clases "logo photo".

```
// Creamos las URL de destino, y del logo.
if (common_logged_in()) {
    $cur = common_current_user();
    $url = common_local_url('all', array('nickname' => $cur->nickname));
} else {
    $url = common_local_url('public');
}
$path = $this->path('css/logo.png');

// Agregamos el logo al Nav.
$action->out->elementStart('li', array('id' => 'liLogoPlugin'));
$action->out->elementStart('address', array('class' => 'vcard'));
$action->out->elementStart('a', array('class' => 'url home bookmark', 'href' => $url));
$action->out->element('img', array('id' => 'logoNav', 'class' => 'logo photo', 'alt' => 'Bolotweet 2.0', 'src' => $path));
$action->out->elementEnd('a');
$action->out->elementEnd('address');
$action->out->elementEnd('li');
```

Como se puede ver en el código anterior, se han creado dos URL's distintas dependiendo de si el usuario está o no logueado. Por otro lado, se han creado todos los elementos con la misma estructura que el logo principal, y todos ellos dentro de un elemento <li>, que formará un nuevo elemento en la lista del *aside* izquierdo, en este caso el primero.

Para generar la url a la imagen, se ha optado por utilizar una URL relativa, de manera que se ha colocado el logo en la carpeta *css* del plugin, y se ha generado la dirección a través de la función *path* (proporcionada por la clase *plugin)*.

Con este código, el resultado final es el siguiente:



Modificar apariencia del logo y eliminar el efecto hover del ratón

Ficheros modificados: logo.css

A pesar de que el logo se ha situado donde se quería, se ha decidido mejorar su aspecto y su comportamiento. Por un lado, cuando se sitúa el ratón encima, se muestra una raya horizontal azul al comienzo de la foto.

Para solucionar esto, y tras muchas pruebas con el inspeccionador de elementos del navegador, se pudo ver como este enlace estaba heredando una propiedad *hover* que provocaba poner el background en azul al situar el ratón encima.

Esto se produce porque hay una regla que indica que todo enlace <a> situado dentro del bloque #site_nav_local_views, que es el ID del aside izquierdo, se le aplique automáticamente una serie de reglas para colocar el *background* azulado. Esto es porque como en este *aside* se espera que los enlaces sean nuevos *menu-items*, es correcto que se les aplique esta propiedad *hover*.

Sin embargo, en este caso esto no es deseable, por lo que se ha tenido que añadir una regla CSS en la que se elimine el *background* para este elemento.

```
#nav_local_default .vcard a:hover {
      background: none;
}
```

Como se puede ver en la regla, se ha indicado que el *background* al pasar el ratón por encima en los enlaces que estén dentro de una clase *vcard*, que en este caso únicamente la tiene el logo, y que además estén dentro de la lista *nav_local_default*.

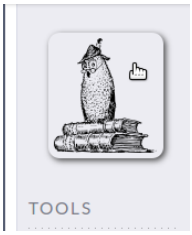
De esta manera, manteniendo el resto de reglas, se ha conseguido que esa raya no se muestre.

Por otro lado, en cuanto a la apariencia del logo, también se le han aplicado una serie de reglas extras para mejorar su visualización.

Lo primero que se ha hecho es centrarlo en el *aside*, redondearle las esquinas, y proporcionarle un poco de *padding* para separar la imagen de su propio borde. Posteriormente se le ha añadido un borde fino, sombra paralela, y un poco de margen inferior para separarlo del menú Tools.

```
#logoNav {
    margin-left: 8px;
    padding: 5px;
    border-radius: 10px;
    box-shadow: 2px 2px 5px rgb(66, 66, 66);
    border: 0px solid rgb(220, 220, 220);
    margin-bottom: 30px;
}
```

El resultado final es el siguiente:



El logo no se muestra en la pantalla de login

Ficheros modificados: LogoPlugin.php

A pesar de que una vez logueado en BoloTweet el nuevo logo se mostraba correctamente, en la pantalla principal de login, el logo no se mostraba.

El problema es ocasionado porque el hook utilizado únicamente se ejecuta dentro de BoloTweet. Para solucionar esto, se ha tenido que buscar y utilizar un nuevo hook que permitiera añadir de la misma manera el logo, pero en la pantalla de login.

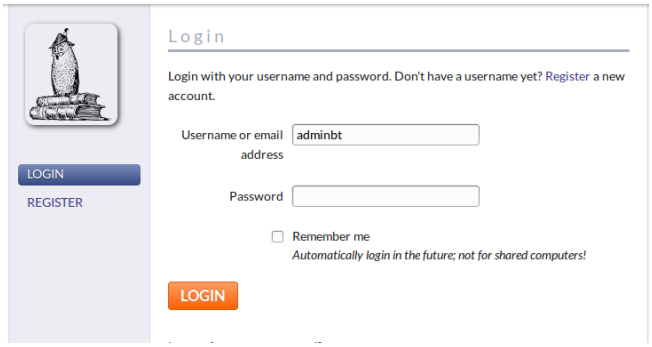
Finalmente se ha decidido utilizar el hook StartLoginGroupNav, que permite insertar elementos en el aside pero de la pantalla de Register y Login.

Para incluir el logo en este lugar, se ha tenido que crear de nuevo en el fichero *LogoPlugin.php* una función que utilice este evento, onStartLoginGroupNav.

En este caso, como solamente hay una pantalla, se ha creado el logo sin enlace. Por tanto únicamente se he creado el elemento <li>, y el elemento <img>, eliminando las comprobaciones para generar las URL de redirección, y los elementos <a> y <address>.

```
function onStartLoginGroupNav($action) {
        $path = $this->path('css/logo.png');

        $action->elementStart('li', array('id' => 'liLogoPlugin'));
        $action->element('img', array('id' => 'logoNav', 'class' => 'logo photo',
        'alt' => 'Bolotweet 2.0', 'src' => $path));
        $action->elementEnd('li');

        return true;
}
```

Como la imagen tiene el mismo ID que en la otra función se le van a aplicar las mismas reglas CSS, y el resultado por tanto será el correcto.



Eliminar el logo del búho en la versión móvil

Ficheros modificados: mp-screen.css

Tras diversas pruebas utilizando el nuevo logo, se vio la necesidad de no mostrarlo en la versión móvil, ya que ocupa demasiado espacio para el tamaño de las pantallas de estos dispositivos.

Tal y como se explicó en otras tareas, la visualización de los dispositivos móviles se controla a través de un plugin de StatusNet denominado MobileProfile. Es éste plugin quien controla cuándo el

dispositivo actual es un dispositivo móvil, y se encarga de aplicar una serie de reglas predefinidas para cambiar la apariencia de la página.

Todas estas reglas están definidas en el fichero mp-*screen.css*, por lo que es ahí donde se han añadido las reglas para ocultar el logo. En este caso es imposible realizar esta acción desde el propio plugin *LogoPlugin*, ya que al no disponer de ningún método ni de ninguna forma para detectar dispositivos móviles, no se puede realizar ninguna distinción.

La única posibilidad que ofrece StatusNet para trabajar sobre la versión móvil es a través del plugin MobileProfile, por lo que es ahí donde se debe añadir todo el comportamiento nuevo que se quiera.

En este caso, lo que se ha hecho es aprovechar el ID del elemento <li> que se crea al insertar el logo, e indicar a través de una regla CSS que no se muestre. De esta manera, independientemente de que se esté en la página de login, o ya dentro del sistema, en ningún caso se mostrar ese elemento <li>, y por tanto tampoco la imagen que está dentro del mismo.

```
li#liLogoPlugin {
display: none;
}
```

Como se puede ver en la siguiente imagen, en ningún caso se muestra el logo.

GuiaPlugin

La idea de este plugin es ofrecer un apartado en BoloTweet para poder descargar el Manual de uso.

Crear la estructura interna del plugin

Ficheros modificados: GuiaPlugin.php, guia.css

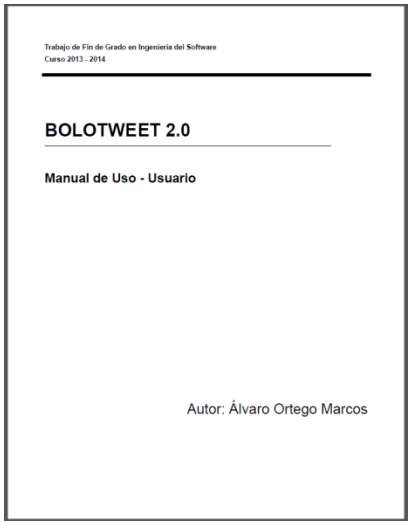
Antes de comenzar a explicar la implementación del Plugin, es necesario crear toda la estructura de archivos y carpetas que se vayan a necesitar. En este caso se ha creado el fichero principal del plugin GuiaPlugin.php, y las carpetas *actions*, *css*, *files* y *lib.*

Desarrollar el Manual de Uso

Ficheros modificados: Manualv1.pdf

Antes de comenzar a explicar todo el comportamiento de descarga del manual, ha sido necesario desarrollarlo. Se ha generado un fichero PDF con información acerca de todas las opciones principales de BoloTweet, así como sobre su utilización.

Se ha acompañado de imágenes y ejemplos que favorezcan el entendimiento del sistema. También se ha explicado la metodología de trabajo propuesta, así como las herramientas que se han desarrollado para facilitar la realización de esta metodología.


Trabajo de Fin de Grado en Ingeniería del Software
Curso 2013 - 2014

BOLOTWEET 2.0

Manual de Uso - Usuario

Autor: Álvaro Ortego Marcos

Crear un enlace al manual en la página

Ficheros modificados: GuiaPlugin.php

Tras barajar distintas posibilidades, principalmente entre en el menú "Herramientas", o bien en la barra superior, se ha decidido colocar en esta última, al lado del enlace al "Grade Reports".

Para colocarlo ahí, se ha utilizado el evento StartPrimaryNav, por lo que se ha tenido que crear en el fichero principal del plugin una función que utilice tal evento, onStartPrimaryNav.

Se ha tenido también que crear la función onRouterInitialized, para enlazar la acción asociada a este enlace con una URL, y como en el resto de ocasiones, todos los nuevos ficheros se han tenido que incluir en la función *onAutoLoad*.

La idea es que a través de este enlace, se muestre una página en la que se ofrezca una breve explicación acerca del manual, y un botón desde el cual descargarlo.

Por ello, en esta función tan solo debe crearse un menu_item cuya acción asociada sea la encargada de generar esta página.

```
function onStartPrimaryNav($action) {
        $user = common_current_user();
        if (!empty($user)) {
            $action->menuItem(common_local_url('guiamostrar'), _m('Manual de Uso'), _m('Guía de uso BoloTweet'), false, 'nav_guia');
}
        return true;
    }
```

El resultado final es el siguiente:

Crear la acción que reciba el enlace "Manual de Uso"

Ficheros modificados: GuiaPlugin.php, GuiamostrarAction, guia.css, GuiaForm.php

Para poder generar una página con información y un botón de descarga, es necesario desarrollar una acción que realice todo este proceso.

En este caso se ha creado el fichero *GuiamostrarAction.php*, y se ha incluido en la función onAutoLoad de GuiaPlugin.

Como esta acción no debe realizar ningún tipo de proceso más allá de mostrar la nueva página, todo la lógica se ha implementado en la función showContent, que es la función que es necesario sobrescribir para sustituir la parte central de la página, y a la que se llama desde la función showPage, que en este

caso se llama desde la función *handle* de la acción.

En la función showContent, lo que se ha hecho es añadir una serie de párrafos con instrucciones, y por último se ha mostrado el formulario que va a permitir vincular la acción final que genere la descarga del Manual.

Este formulario, representado por el fichero *GuiaForm.php*, está formado nada más que por un botón de tipo submit, que será a través del cual se llame a la última acción, y por un elemento *hidden* a través del cual se le pasará el nombre del fichero a esta última acción.

```
$this->out->hidden('filename-guide', $this->fileName, 'filename');
$this->out->submit('guide-download-submit', 'Descargar', 'submit', $this->fileName, 'Descarga la Guía');
```

Como se puede observar en el código, el elemento *hidden* se rellena con el valor del atributo de clase $this->fileName, definido anteriormente en el constructor. Es por ello, por lo que en la creación de este formulario va a ser necesario indicarle el nombre del fichero como segundo parámetro.

El código de la función showContent es el siguiente:

```
        $this->elementStart('p');
        $this->raw("En este manual podrás conocer las principales funcionalidades y mejoras de BoloTweet. "
                . "Aprenderás cómo manejar algunas de sus opciones, así como otros aspectos importantes de"
                . "su uso.<br/><br/>Esta guía estará constantemente actualizada a medida que se vayan añadiendo novedades "
. "al sistema.");
        $this->elementEnd('p');
        $this->element('br');
        $this->element('br');
        $this->elementStart('p', array('class' => 'text-download-guide'));
$this->raw('Descarga la Guía en Formato PDF    [Versión v1.0]');
$this->elementEnd('p');
        $downloadBut = new GuiaForm($this, "Manualv1.pdf");
$downloadBut->show();
    }
```

Añadir mecanismo de descarga del manual

Ficheros modificados: GuiaPlugin.php, GuiadescargarAction

El último paso necesario en este plugin es la creación del mecanismo final de descarga, en este caso a través de la acción *GuiadescargarAction*, a la cual se llama desde el formulario *GuiaForm*.

Esta acción no necesita mostrar ninguna nueva página, por lo que todo el mecanismo se va a realizar dentro de la función *handle*, y en ningún caso se va a llamar a la función *showPage*.

Lo primero que se ha hecho es realizar una comprobación para permitir solamente la descarga a usuarios registrados y logueados.

```
if (!common_logged_in()) {
            $this->clientError(_('Not logged in.'));
return;
        }
```

El siguiente paso ha sido buscar una forma correcta de ofrecer la descarga de un archivo local a un usuario, con el mayor número de comprobaciones, y con un comportamiento correcto en todos los casos.

En casi todos los casos se recomienda utilizar la función readfile, de PHP, junto con una serie de cabeceras que le indiquen al navegador qué es ese fichero, y para que sepa que debe generar la descarga de ese fichero.

Una vez conocido lo fundamental de este mecanismo, se ha pasado a implementarlo.

Lo primero que se ha hecho es obtener el nombre del fichero que se ha recibido a través del elemento hidden del formulario.

Una vez conocido el nombre, se han generado dos URL's, una local, y otra remota, a las que se ha añadido el nombre del fichero. La local se refiere a la estructura interna de carpetas, y se va a utilizar para comprobar la existencia del fichero de descarga.

La segunda URL se va a utilizar para generar la descarga del fichero a través del navegador.

```
$url = common_path() . 'local/plugins/Guia/files/';
$dir = substr($_SERVER['SCRIPT_FILENAME'], 0, -9) . 'local/plugins/Guia/files/';
$pathURL = $url . $file;
$pathDIR = $dir . $file;
```

Algunos detalles importantes a comentar, es que para poder generar un Plugin compatible con cualquier instalación de BoloTweet, y provocado por el diferente nombre entre la instalación de Producción y Desarrollo, se ha utilizado la función *common_path*, para generar la ruta remota, ya que siempre proporciona la base de la URL correcta. Por otro lado, para generar la URL local de manera dinámica, lo que se ha tenido que hacer es utilizar una variable de servidor, en este caso *SCRIPT_FILENAME*, en la que se incluye la ruta de la instalación local.

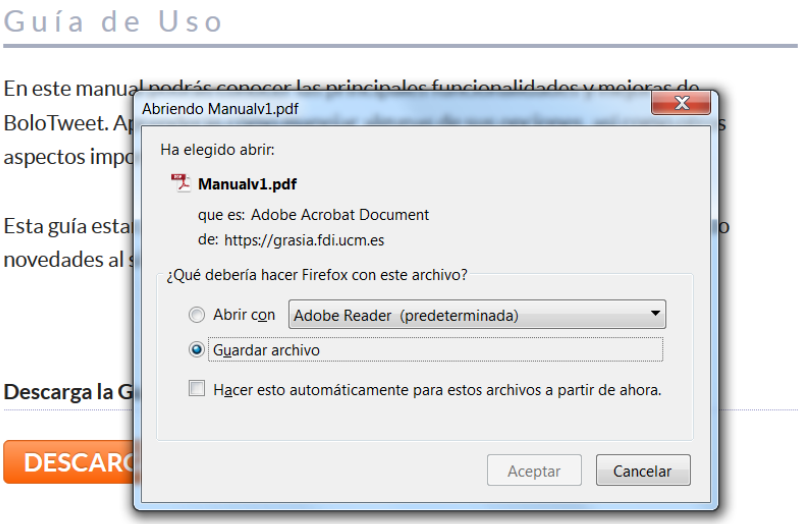
De esta manera, tan sólo es necesario utilizar la función substr para eliminar la parte correspondiente al script, y de esta manera se obtendría la ruta local dinámica, que servirá de igual manera en cualquier instalación de BoloTweet.

Una vez generadas ambas rutas, se les añade el nombre del fichero, y se comprueba que el fichero recibido exista, a través de la función *is_file* de PHP.

Si el fichero no existe, se abortará la ejecución. Si por el contrario el fichero existe y es accesible, se indicarán las cabeceras necesarias para indicarle al navegador el tipo de archivo, y para indicarle que debe generar la descarga, y por último se llamara a la función readfile, que será la que provoque la generación de la descarga:

```
if (is_file($pathDIR)) {
   // Definir headers
    header("Content-type: application/pdf");
    header("Content-Transfer-Encoding: Binary");
    header("Content-disposition: attachment; filename=\"" . $file . "\"");
    readfile($pathURL);
} else {
            die("El archivo no existe.");
        }
    }
```

Es importante mencionar que a pesar de ejecutarse estos últimos pasos sobre una acción nueva, con una URL asociada, no se produce ninguna redirección, ni se ninguna página en blanco, tan sólo se genere la descarga, y se queda en la misma página, por lo que el comportamiento es el esperado.



Modificar visualización del formulario

Ficheros modificados: guia.css

Una vez terminado todo el mecanismo de descarga del manual, se han aplicado una serie de reglas sobre el formulario de manera que tenga una mejor apariencia.

Todas estas reglas se han incluido en el fichero guía.css, vinculado a través de la función *onEndShowStyles* del fichero principal del plugin *GuiaPlugin.php*.

Se han añadido reglas para modificar el título de la descarga, poniéndolo con un subrayado y en negrita.

```
.text-download-guide{
    border-bottom:1px dotted #9999CC;
font-weight: bold;
}
```

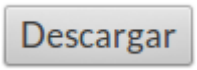
Por otro lado, también se ha modificado el botón de descarga, aplicándole una apariencia más acorde con BoloTweet.

En este caso se han aplicado modificaciones de bordes, colores, *background*, degradados… por lo que las reglas son ligeramente más complejas.

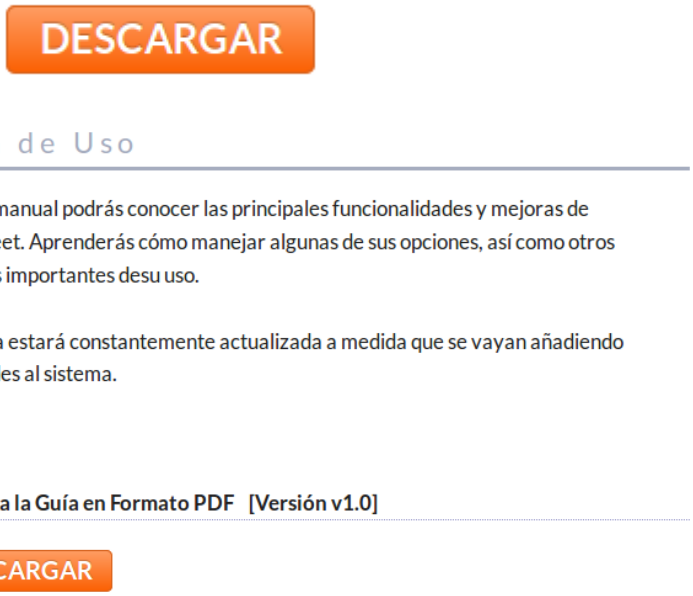
```
height: 1.7em;
    padding: 0px 10px;
    color:#fff;
    font-weight: bold;
    text-transform: uppercase;
    font-size: 1.2em;
    text-shadow: 0px -1px 0px rgba(0, 0, 0, 0.2);
    border: 1px solid #d7621c;
    border-radius: 4px;
    -moz-border-radius: 4px;
    -webkit-border-radius: 4px;
    background: #FB6104;
    background: -moz-linear-gradient(top, #ff9d63 0%, #fb6104 100%);
    background: -webkit-gradient(linear, left top, left bottom, color-
stop(0%,#ff9d63), color-stop(100%,#fb6104));
    background: -webkit-linear-gradient(top, #ff9d63 0%,#fb6104 100%);
    background: -o-linear-gradient(top, #ff9d63 0%,#fb6104 100%);
    background: -ms-linear-gradient(top, #ff9d63 0%,#fb6104 100%);
    filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#ff9d63',
endColorstr='#fb6104',GradientType=0 );
    background: linear-gradient(top, #ff9d63 0%,#fb6104 100%);
```

El resultado final es el siguiente:

Antes:



- Después:



TaskPlugin

La idea de este plugin es ofrecer un sistema de "tareas", en el que el profesor pueda iniciar un evento, y a los alumnos asociados a ese evento se les genere una tarea pendiente. El concepto de "tarea" no es más que una ayuda para los alumnos, a través de la cual puedan generar los tweets de cada clase con mayor facilidad, ya que en este caso, los tags ya vienen incluidos, y la selección del grupo ya está hecho.

Este plugin ha sido desarrollado en vista de la cantidad de errores tipográficos producidos al escribir la selección de grupo o las etiquetas vinculadas a un tweet.

Por otro lado, este sistema de gestión de tareas también servirá como ayuda para el profesor, quien podrá llevar un control sobre las tareas iniciadas, así como el número de alumnos que la han completado.

Crear estructura para el plugin

Ficheros modificados: TaskPlugin. Php, task.css, task.js

El primer paso ha sido crear toda la estructura de archivos y carpetas que han sido necesarias para la implementación de TaskPlugin. En este caso, se ha creado el archivo principal TaskPlugin. Php, y las siguientes carpetas:

- actions
- classes
- lib
- js
- scripts
- css

Crear menuItem debajo de Apuntes

Ficheros modificados: TaskPlugin. Php

El siguiente paso, tras buscar la mejor posición para incluir un enlace a las tareas, ha sido colocar un elemento de menú en la sección herramientas, justo después de "Apuntes".

Sin embargo, como esta sección de "Herramientas" se había creado *ad hoc* en el plugin NotesPDF, utilizando el *hook startDefaultLocalNav*, era imposible añadir un elemento dentro de la lista creada en el plugin NotesPDF.

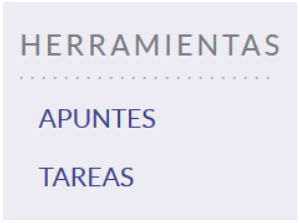
La única opción válida habría sido mantener la creación del menú en NotesPDF, y añadir allí un *hook* para insertar nuevos elementos en la lista. Sin embargo, no es una buena opción el crear dependencias entre plugin, ya que si en algún momento se desactiva NotesPDF, Task también dejaría de funcionar.

Por ello, la solución ha sido mover toda la parte de creación de esa sección al *core* de BoloTweet, añadir allí un nuevo *hook*, y utilizarlo tanto en NotesPDF como en Task para insertar elementos en la sección herramientas.

La parte de creación de la nueva sección en el estructura interna de BoloTweet ha sido explicada con detalle en la tarea *Crear nuevos hooks onStartToolsLocalNav y onEndToolsLocalNav*.

En este caso, para utilizar el nuevo hook, se ha tenido que crear una función que lo utilice, en este caso *onStartToolsLocalNav*.

```
$actionName = $action->trimmed('action');
        $user = common_current_user();
        if (!empty($user)) {
      $action->menuItem(common_local_url(taskcreate), _m('Tareas'), _m('Tareas de
      Clase'), $actionName === 'taskcreate, 'nav_task');
}
```



Crear clase/s para el almacenamiento interno

Ficheros modificados: TaskPlugin. Php, Task.php, Task_Grader.php

El siguiente paso ha sido diseñar la estructura interna de este Plugin. Como va a ser necesario que se vinculen tareas a usuarios, y que además se lleve un histórico de las mismas, va a ser necesario crear alguna tabla en la base de datos.

Finalmente, tras pensar y diseñar su estructura, se ha decidido crear 2 tablas con el siguiente formato:

- Task_Grader
  - **ID**
  - graderID
  - groupID
  - tag
  - status
  - cdate

- Task
  - **ID** (foreign key a ID de Task_Grader)
  - **userID** (alumno)
  - status

La idea ha sido la siguiente. Primero se ha creado una tabla *task_grader* en la que se almacenarán las tareas creadas por los profesores. De esta manera, cuando se cree una nueva tarea, va a ser necesario almacenar un ID único vinculado a cada nuevo registro de la tabla. Posteriormente el ID del profesor, el ID del grupo, la fecha en que se ha creado, el estado de la misma, y posteriormente un campo *tag* opcional.

La otra tabla, Task, se utilizará para vincular alumnos con tareas. Como cada alumno puede o no, contestar una tarea, es necesario llevar un control individual. Para ello, lo que se ha hecho es incluir en cada nuevo registro un ID, que hará referencia al ID de la tarea creada por el profesor, el ID del usuario, y por último el estado de la tarea.

Con todos estos datos está completamente cubierto el comportamiento del plugin, pudiendo hacer en cualquier momento un *join* entre ambas tablas para obtener la información que se necesite.

En cuanto a las claves primarias, en la tabla Task_Grader se ha decidido que el campo ID sea la PK. Por otro lado, en la tabla Task, se ha decidido crear una clave primaria múltiple, formada por el campo ID y el campo UserID. La necesidad de esta clave primaria múltiple, viene dada porque si se quisiera colocar como clave primaria únicamente el ID, al existir un registro con el mismo ID de tarea para cada alumno, no se permitiría añadir más de una tarea. De esta manera, el funcionamiento es precisamente el esperado.

Una vez decididas las tablas, el siguiente paso es crear una clase para cada una, con tantos atributos como columnas, y con la definición de la estructura en la función schemaDef.

Es importante que esta función esté implementada sin ningún error, ya que en la primera ejecución del plugin se utilizará para crear cada respectiva tabla.

Para la tabla Task Grader, se ha creado el archivo */classes/Task_Grader.php*. Al ser una clase representante de una tabla, y que se va a comunicar con MySQL, es necesario que herede de *Managed_DataObject.* Se ha creado la función staticGet y multiGet, para poder obtener objetos de esta clase, y por último se ha implementado la función *schemaDef*, en la que se define la estructura a nivel SQL.

```
public static function schemaDef() {
        return array(
            'description' => 'Task Graders',
            'fields' => array(
                'id' => array(
                    'type' => 'serial',
                    'not null' => true,
                    'description' => 'Task ID'
```

```
                ),
                'graderid' => array(
                    'type' => 'int',
                    'not null' => true,
                    'description' => 'ID del grader'
                ),
                'groupid' => array(
                    'type' => 'int',
                    'not null' => true,
                    'description' => 'Group ID'
                ),
                'tag' => array(
                    'type' => 'varchar',
                    'length' => 50,
                    'not null' => true,
                    'description' => 'Task tag'
                ),
                'status' => array(
                    'type' => 'tinyint',
                    'not null' => true,
                    'description' => 'Status of Task'
                ),
                'cdate' => array(
                    'type' => 'date',
                    'not null' => true,
                    'description' => 'Date the task was created'
),
            ),
            'primary key' => array('id'),
        );
    }
```

En cuanto a la elección del tipo de las columnas, únicamente destacar el campo ID, donde se ha optado por el tipo *serial (big int auto-increment)*, y el campo cdate, de tipo *date*, ya que el tipo *timestamp* contiene la hora, y en este caso no se necesita.

Por otro lado, para la implementación de la tabla Task, se ha cread el archivo */classes/Task.php*, con sus funciones staticGet y multiGet, sus atributos de clase, y su función *schemaDef*.

```
    public static function schemaDef() {
            return array(
                'description' => 'Task Graders',
                'fields' => array(
                    'id' => array(
```

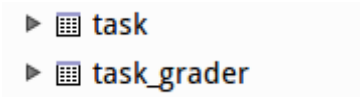
```
                    'type' => 'serial',
                    'not null' => true,
                    'description' => 'Task ID'
                ),
                'userid' => array(
                    'type' => 'int',
                    'not null' => true,
                    'description' => 'ID del grader'
                ),
                'status' => array(
                    'type' => 'tinyint',
                    'not null' => true,
                    'description' => 'Status of Task'
                ),
            ),
            'primary key' => array('id', 'userid'),
    );
        }
```

Una vez creadas ambas clases, el siguiente paso ha sido incluir en el fichero *TaskPlugin. Php* la función onCheckSchema, y añadir en su interior las respectivas funciones *schemaDef*, de manera que se pueda comprobar su estructura cada vez que se ejecute el plugin, y en el caso de que no existan las tablas, esta función se encargará de crearlas.

```
function onCheckSchema() {
        $schema = Schema::get();

        $schema->ensureTable('task', Task::schemaDef());
        $schema->ensureTable('task_grader', Task_Grader::schemaDef());

return true;
    }
```

Por otro lado, como en el resto de casos, al añadir nuevos archivos al plugin, ha sido necesario incluirlos en la función *onAutoLoad*.

Una vez creado todo el comportamiento, se ha ejecutado por primera vez el plugin, y todo ha funcionado correctamente, las tablas se han creado con la estructura esperada.



Añadir número de tareas pendientes en el menú_item

Ficheros modificados: TaskPlugin. Php, task.css

Antes de proseguir con las acciones y la lógica del plugin, se ha decidido añadir una mejora de visualización para los alumnos. En este caso, para que no tengan que entrar en "Tareas", para comprobar o no, si tienen algo pendiente, se ha decidido añadir en el propio menú_item del *aside* izquierdo, un pequeño espacio con el número de tareas pendientes.

Sin embargo, tal y como se había creado el menú_item en la función *onStartToolsLocalNav,* esto no va a ser posible, ya que se ha utilizado la función menú_item, y no es posible personalizarla.

Por ello, se ha tenido que ver el funcionamiento interno de esta función, para crear manualmente el enlace de menú, y poder añadir un espacio con el número de tareas pendientes.
Es importante recrear todo el comportamiento de la función menú_item, pues además del enlace, hay que realizar una comprobación para saber si marcar como seleccionado o no, el enlace.

Además, como los profesores no van a tener tareas pendientes, también va a ser necesario añadir una comprobación para únicamente mostrarlas a alumnos.

Finalmente, el código de la función onStartToolsLocalNav ha quedado de la siguiente manera:

```
$actionName = $action->trimmed('action');

        $user = common_current_user();
        if (!empty($user)) {

            if ($actionName === 'taskcreate')
                $action->elementStart('li',  array('id'  =>  'nav_task',  'class'  =>
'current'));
            else
                $action->elementStart('li', array('id' => 'nav_task'));

            $action->elementStart('a',                array('href'                =>
common_local_url('taskcreate'), 'title' => _m('Tareas de Clase')));
$action->text('Tareas');

            // Si es alumno, mostramos las tareas pendientes.
            if (!$user->hasRole('grader')) {

                // Llamamos a la función que obtenga el numero de tareas
$number = Task::getNumberPendingTasks($user->id);

if ($number != 0)
$action->element('span', 'pending-tasks-number', $number);
```

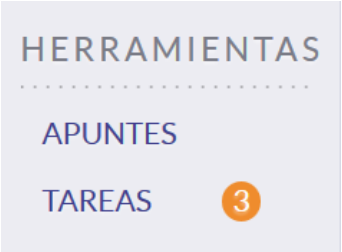
```
            }

            $action->elementEnd('a');
            $action->elementEnd('li');
}
```

Mencionar algunos aspectos importantes de la implementación. Como se puede ver, se ha tenido que crear una comprobación de manera que si el nombre de la acción coincide con *'taskcreate'*, se cree el elemento <li> con la clase current, para que se le apliquen las reglas de elemento seleccionado. Por otro lado, se ha realizado la comprobación de que el alumno no sea profesor, y por último se ha hecho uso de la función Task::getNumberPendingTasks, que posteriormente debe implementarse, y para insertar el número de tareas se ha utilizado un elemento de tipo .

Todo ello ha sido modificado a través de reglas CSS en el fichero *task.css*.

```
.pending-tasks-number{
    margin-left: 30px;
    border-radius: 10px;
    background-color: #EF8C2D;
    padding: 1px 5px;
    color: #FFF;
}
```

Y el resultado final ha sido el siguiente:



Implementación de la función getNumberPendindTasks

Ficheros modificados: Task.php

Como para obtener el número de tareas pendientes en el menú ítem se ha utilizado esta función, ha sido necesario implementar su comportamiento.

Para ello, hay que recordar que las tareas vinculadas a cada alumno tiene un campo status. Este campo puede contener 3 tipos de valores distintos:

-1 → Tarea rechazada

0 → Tarea pendiente

1 → Tarea completada

De esta manera, ha sido sencillo implementar una consulta que localice el número de tareas pendientes vinculadas a un usuario:

```
$qry = 'select count(t.id) as number '
                . 'from task t '
                . 'where t.userid = ' . $userid
                . ' and t.status = 0';
```

Crear formulario para iniciar una tarea

Ficheros modificados: TaskPlugin. Php, InitForm.php

Cada vez que un profesor acceda a "Tareas", podrá visualizar los grupos a los que pertenece, y al lado de cada uno de ellos un botón para iniciar la tarea.

En este caso, el formulario para iniciar la tarea se ha definido en el fichero *InitForm.php*.

Este formulario recibe como parámetros extra el *groupid* y un objeto *result*. El ID del grupo es necesario para poder pasárselo a la acción asociada al enviar el formulario. En cuanto al objeto *result*, se utiliza para crear el botón de una manera u otra.

Este parámetro es el resultado de la función *checkTask*, que comprueba si una tarea para un grupo ya está iniciada. De esta manera, dependiendo de este parámetro, el formulario se creará activo o deshabilitado.
El formulario contiene en cualquier caso tres elementos *hidden*, con el *groupid*, *taskid* y *status*, obteniéndose estos dos últimos a través del objeto result.

Posteriormente, se comprobará si el elemento status es distinto de 1. Si lo es, quiere decir que la tarea aún no ha sido iniciada, y por tanto es necesario crear el botón de Iniciar Tarea activo, y un elemento de tipo input para insertar un tag opcional relacionado con la tarea.

```
$this->out->element('input', array('type' => 'submit',
                'id' => 'task-submit-' . $this->groupid,
                'class' => 'submit task-button-enabled',
'value' => 'Iniciar',
                'title' => 'Crea una tarea para este grupo')),
             'onclick' => 'updateHistorical(' . $this->user->id . ',' . $this-
             >groupid . ')'));;


            $this->out->element('input', array('type' => 'text',
                'name' => 'task-tag-' . $this->groupid,
'class' => 'task-tag',
                'maxlength' => "13",
                'title' => 'Añade un tag relacionado con la tarea'));

$this->out->element('p', 'label-for-tag', 'Tag: (Opcional)');
```

Si por el contrario el status recibido es igual a 1, se creará un botón deshabilitado en el que se indicará que la tarea de ese día ya está iniciada.

```
$this->out->element('input', array('type' => 'button',
                'class' => 'task-disabled',
                'value' => 'Iniciada',
'title' => 'La tarea ya está iniciada.',
                'disabled' => 'disabled'));
```

Este formulario tiene como acción asociada *taskcreate*, y es importante mencionar que se le ha asignado la clase "Ajax", para posteriormente en la acción recargarlo sin necesidad de refrescar la página.

Crear formulario NoticeTaskForm

Ficheros modificados: TaskPlugin. Php, NoticeTaskForm.php

En la parte de los alumnos, va a ser necesario crear un espacio donde puedan escribir la tarea. Como esta caja va a tener una personalización y un comportamiento diferente a la ya existente al comienzo de la página, ha sido necesario crear un formulario personalizado.

Este formulario se ha implementado en el archivo *NoticeTaskForm.php*, y está basado en el formulario original de anotaciones, *noticeform.php*.

Diferencias y modificaciones respecto al original:

- Se han incluido los atributos de clase *msg* y *taskid*. El campo *msg* se ha utilizado para mostrar un mensaje de error en el caso de que el tratamiento de la noticia por la acción produzca alguna excepción.
- Se ha modificado el constructor para incluir estos dos nuevos parámetros.

```
function __construct($action, $taskid, $options = null, $msg = null) { … }
```

- Modificado ID, clase y acción asociada (*newnotice)*.
- Añadida un nuevo elemento <p> al formulario en el caso de que el atributo *msg* no esté vacío. Este elemento servirá como mecanismo de errores.

```
        if (!empty($this->msg)) {
            $this->out->element('p', array('class' => 'error error-notice-
task'), $this->msg);
        }
```

- Comprobación para crear el contador de caracteres:

```
if ($contentLimit > 0) {

            $count = $contentLimit;
            if ($this->content) {
                $count = $contentLimit - mb_strlen($this->content);
            }
```

```
            if ($count < 0) {
             $this->out->element('span', array('class' => 'count count-negative'),
             $count);
            } else
                $this->out->element('span', array('class' => 'count'), $count);
}
```

- Añadidos nuevos elementos hidden con el groupid y el taskid.
- Eliminada parte de geolocalización.
- Añadido el selector de grupo precargado únicamente con el grupo que corresponda.

```
$this->out->elementStart('div', 'to-selector');
$dropdown[$this->to_group->id] = $this->to_group->getBestName();
$this->out->dropdown('notice_to', 'Para:', $dropdown, null, false,
$this->to_group->getBestName());
$this->out->elementEnd('div');
```

- Añadido evento *onclick* al botón *submit* para reducir el número de tareas pendientes al enviar.

Crear acción que reciba el enlace Tareas

Ficheros modificados: TaskPlugin. Php, TaskcreateAction.php, task.css, GradesGroup.php, Task_Grader.php, Task.php

El siguiente paso ha sido crear toda la lógica de tareas, tanto para la parte de profesor, como la parte de alumno.

En este caso, y para no crear archivos extra innecesarios, se ha realizado todo a través del fichero *TaskcreateAction.php*.

Este fichero se ha tenido que incluir en la función *onAutoLoad*, y en la función *onRouterInitialized*, para vincular la acción con una URL.

Como esta acción también trata todos los eventos de formularios relacionados con las tareas, en este apartado únicamente se explicará la parte relativa a lo que se muestra al pulsar el menú "Tareas", tanto para un alumno como para un profesor.
La función que se encarga de sobrescribir el contenido central de la página es *showContent*. Como en este caso la página a mostrar será diferente para alumnos y para profesores, se ha tenido que realizar una distinción:

- Si el usuario es *grader*, se va a ejecutar la función *createTask,* la que le permitirá al profesor poder iniciar tareas para cualquier de sus grupos.
- Si el usuario no es *grader*, se ejecutará la función *showTasks*, que mostrará las tareas pendientes y permitirá contestarlas.

A continuación, se explicará la implementación del método *createTask*.

Como esta función debe proporcionar a cada *grader* un botón para crear una tarea en cualquier de sus

grupos asociados, lo primero que va a ser necesario es desarrollar una función que obtenga los grupos vinculados a un determinado profesor.

Se ha creado por tanto la función *getGroups*, en el fichero *GradesGroup.php*, cuyo contenido es el siguiente:

```
$qry = 'SELECT gg.groupid'
                . ' FROM grades_group gg'
                . ' where gg.userid=' . $graderid;
```

Una vez obtenidos los IDs de los grupos, se ha utilizado el método multiGet para obtenerlos todos, y posteriormente se han ido recorriendo uno por uno a través de un bucle *foreach*.

```
$groupsIds = Gradesgroup::getGroups($this->user->id);
$groups = User_group::multiGet('id', $groupsIds)->fetchAll();
foreach ($groups as $group) { … }
```

Por cada grupo, lo primero que se ha comprobado es si ya tiene una tarea iniciada. Esto es un detalle importante, ya que un profesor sólamemente puede crear una tarea al día para un mismo grupo.

```
$result = Task_Grader::checkTask($this->user->id, $group->id);
```

La función checkTask utiliza el userID, el groupID, y la función de SQL *CURDATE* para comprobar si ya hay una tarea iniciada ese mismo día.

```
$qry = 'select tg.status as status, tg.id as id '
                . 'from task_grader tg '
                . 'where tg.graderid =  "' . $graderid . '"'
                . ' and tg.groupid = ' . $groupid
                . ' and tg.cdate = CURDATE()';
```

El siguiente paso, es crear el formulario *InitForm*, explicado en la tarea *Crear formulario para iniciar una tarea*, pasándole como parámetro el resultado de la función anterior.

```
  // Creamos el form para ese grupo.
$form = new InitForm($this, $group->id, $result);
$form->show();
```

De esta manera, se podrá visualizar una página en la que se mostrará cada grupo junto con un botón y un campo de texto en el caso de que la tarea no haya sido iniciada, o bien un botón deshabilitado si la tarea ya se ha iniciado.

El comportamiento de TaskCreate al pulsar en *Iniciar* ha sido explicado en la tarea *Crear función initTask*.

A continuación solamente queda incluir el histórico de tareas para ese grupo, a través del cual se podrá

ver un listado de las tareas vinculadas con el grupo, así como información sobre las mismas. Desde este histórico, los profesores podrán cancelar y reabrir las tareas.

El funcionamiento de esta parte está explicado en la tarea *Crear histórico de tareas*.

En cuanto al apartado de Tareas en el alumno, la función la función que se ha implementado es *showTasks*.

En este caso, la idea es que se muestre una página con las tareas pendientes, ofreciendo la posibilidad de rechazarla, o completarla. Si se decide completar, se mostrará la caja de texto para incluir la anotación, y se enviará. Si se rechaza, se solicitará confirmación y se eliminará.

Para ello, el primero paso es comprobar a través de la función *getPendingTasks* las tareas pendientes del usuario.

```
$taskIds = Task::getPendingTasks($this->user->id);
```

Esta función únicamente devuelve los ID's, por lo que posteriormente ha sido necesario utilizar la función *multiGet* de Task para obtener todas las tareas:

```
$qry = 'select t.id as taskid '
                . 'from task t '
                . 'where t.userid = ' . $userid
                . ' and t.status = 0';
```

Si el array devuelto por la función es vacío, se informará de que no tiene ninguna tarea pendiente.

En el caso de sí tener, se irán recorriendo una a una a través de un bucle *foreach*.

Para cada tarea, se muestra el nombre de la asignatura, la fecha, y se crean dos botones con eventos *JavaScript* asociados, que permitirán rechazarla o completarla.

```
$this->elementStart('div', array('id' => 'options-task-' . $task->id, 'class' =>
'options-task'));

                $this->element('input', array('type' => 'button', 'class' =>
'button-option-reject', 'value' => 'Rechazar', 'onclick' => 'showConfirmDialog(' .
$task->id . ');'));

                $this->element('input', array('type' => 'button', 'class' =>
'button-option-complete', 'value' => 'Completar', 'onclick' => 'mostrarBox(' .
$task->id . ');'));

$this->elementEnd('div');
```

El funcionamiento de los eventos está explicado en las tareas *Crear función showConfirmDialog* y *Crear función mostrarBox*.

Tras crear estos botones, se crea un cuadro de confirmación de rechazo, y la caja para incluir la anotación, pero ambos elementos ocultos, en espera de la acción del usuario.

// Cuadro de confirmación de rechazo

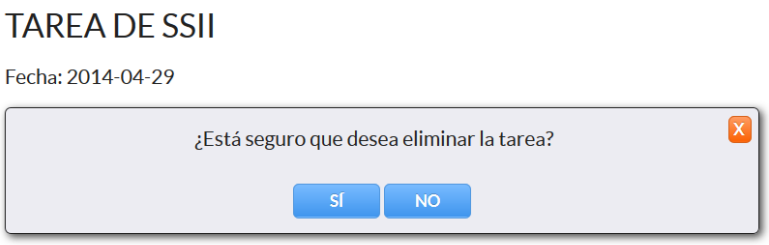
```
$this->elementStart('div', array('id' => 'confirm-reject-task-' . $task->id,
'class' => 'confirm-reject-task'));

                $this->element('a', array('class' => 'close-confirm-dialog', 'href'
=> 'javascript:hideConfirmDialog(' . $task->id . ');'), 'x');

                $this->element('p', null, '¿Está seguro que desea eliminar la
tarea?');

$this->elementStart('form', array('action' => common_local_url('taskcreate'),
'method' => 'POST', 'class' => 'ajax'));

                $this->hidden('hidden-' . $task->id, $task->id, 'delete');

                $this->element('input', array('type' => 'submit', 'value' => 'Sí',
'class' => 'button-option-dialog', 'onclick' => 'deleteTask(' . $task->id . ');'));

                $this->element('input', array('type' => 'button', 'value' => 'No',
'onclick' => 'hideConfirmDialog(' . $task->id . ');', 'class' => 'button-option-
dialog'));

                $this->elementEnd('form');

$this->elementEnd('div');
```

// Caja de anotación

```
$this->elementStart('div', array('class' => 'input_form'));


                if ($task->tag == "") {

                    $notice_form = new NoticeTaskForm($this, $task->id,
array('content' => '#' . $task->cdate, 'to_group' => $group));

                } else {
```

```
                    $notice_form    =    new    NoticeTaskForm($this,    $task->id,
array('content' => '#' . $task->cdate . ' #' . $task->tag, 'to_group' => $group));
                }
                $notice_form->show();

$this->elementEnd('div');
```

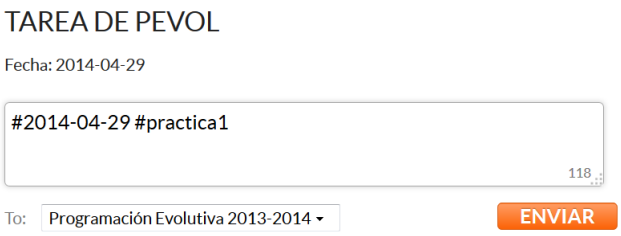
En cuanto al cuadro de confirmación de rechazo, que se mostrará si el usuario pulsa en "Rechazar", está formado principalmente por un formulario con un elemento *hidden* con el ID de la tarea, y un botón *submit* correspondiente a la opción "Sí", y por otro lado un botón con un evento JavaScript correspondiente al "No".



La acción que se ejecuta al pulsar el botón "Sí" está explicada en la tarea *Rechazar una tarea*, y la función JavaScript correspondiente al "No" está explicada en el apartado *Crear función hideConfirmDialog*.

Si se opta por seleccionar "Completar", se mostrará automáticamente el formulario para incluir la noticia, *NoticeTaskForm*. El contenido y el funcionamiento de este formulario se ha explicado en la función *Crear formulario NoticeTaskForm*.

Como se muestra en el código anterior, en el contenido por defecto del cuadro se incluye automáticamente el *tag* con la fecha, y en caso de que exista una etiqueta opcional vinculada, también se añade. Por otro lado, también se le pasa al formulario el grupo de la tarea, para que se cargue en el selector de destino.



Al pulsar enviar, se llama a la acción *NewnoticetaskAction*, la cual se encargará de tratar el contenido de la noticia, validarlo, e insertarla en la base de datos en el caso de que sea correcta.

Este comportamiento está explicado en la tarea *Crear acción para recibir el envío de una tarea*.

Crear función initTask

Ficheros modificados: TaskcreateAction.php, Task_Grader.php, Task.php, Grades.php

Cuando el profesor pulse en el formulario *InitForm*, la acción asociada es de nuevo *TaskCreateAction.php*. Para distinguir en la función *handle* de la acción de qué formulario se viene, se ha realizado una comprobación de manera que si se existe un elemento hidden que se llame *groupid*, el formulario es InitForm, ya que es el único que necesita este valor.

```
if ($this->trimmed('groupid')) {
                $groupid = $this->trimmed('groupid');
                $tag = $this->trimmed('task-tag-' . $groupid);
                $status = $this->trimmed('status');
                $taskid = $this->trimmed('taskid');
                if (substr_compare($tag, "#", 0, 1) == 0) {
                    $tag = substr($tag, 1);
                }

                $this->initTask($groupid, $tag, $status, $taskid);
}
```

Si se entra en la comprobación, se obtienen todos los parámetros del formulario, en este caso el groupid propiamente dicho, el tag de la tarea en caso de que lo lleve, el estado, y el ID de la tarea.

Como es posible iniciar una tarea previamente cancelada, ha sido necesario incluir el estado y el taskid, ya que es posible que la tarea ya estuviera creada.

Una vez obtenidos los datos, se llama a la función *initTask*, que es la que se encarga de crear la tarea, o bien de reiniciarla.

Para conocer si ya estaba o no creada, es tan sencillo como comprobar el estado.

Si es -1, quiere decir que la tarea no existe, por lo que se debe crear.

```
        if ($status == -1) {
            // Registramos la tarea en la tabla task_grader
             $id   =   Task_Grader::register(array('graderid'   =>   $this->user->id,
             'groupid' => $groupid, 'tag' => $tag));


$idsUsers = Grades::getMembersExcludeGradersAndAdmin($groupid);


            // Creamos una tarea asociada para cada alumno del grupo.
if (count($idsUsers) > 0) {
                Task::register(array('id' => $id, 'idsUsers' => $idsUsers));
}
```

```
            $result[0] = 1;
            $result[1] = $id;
        }
```

A continuación se explicará qué se realiza en este código.

Para crear una tarea, debe registrarse en la tabla Task_Grader, de manera que se añada un nuevo registro con el grupo y el profesor asociado, la fecha, el estado (iniciada), y el tag en caso de que se haya definido. A su vez, debe crearse un nuevo registro en la tabla Task para cada alumno del grupo, en el que se indique el ID de la tarea creada, el estado (pendiente) y el id del usuario.

Sin embargo, aquí se produjo un problema, y era el cómo conocer el ID de la tarea recién insertada, ya que el ID es un campo que maneja únicamente MySQL.

Finalmente, tras búsquedas, se descubrió la función LAST_INSERT_ID de MySQL. El problema de esto, es que si durante la creación de la tarea otro profesor registra la suya, pueden producirse comportamientos totalmente erróneos.

Para evitar esto, lo que se ha hecho es lo siguiente:

Se ha creado la función *register*, de Task_grader, que es la que se encarga de registrar la nueva tarea. En esta función se han creado dos consultas, una para insertar el nuevo registro, y otra para obtener el último id insertado.

Además, para evitar que se puedan producir errores con el ID, lo que se ha hecho es realizar las consultas dentro de una transacción, de manera que nadie pueda insertar nada durante ese tiempo, y garantizando así que el ID devuelto es el recién insertado.

Una vez insertado el nuevo registro, el ID obtenido se devuelve, para poder posteriormente utilizarlo al crear las tareas vinculadas a cada alumno.

La función *register* tiene el siguiente aspecto:

```
if ($tag == "") {
            $qry = 'INSERT INTO task_grader (graderid,groupid,cdate,status) '
                    . 'VALUES (' . $graderid . ',' . $groupid . ',CURDATE(), 1)';
        } else {

            $qry = 'INSERT INTO task_grader (graderid,groupid,tag,cdate,status) '
                    . 'VALUES (' . $graderid . ',' . $groupid . ',"' . $tag .
'",CURDATE(), 1)';
        }

        $qry2 = 'SELECT LAST_INSERT_ID() as id FROM task_grader';

        $taskG->query('BEGIN');
        $taskG->query($qry);
        $taskG->query($qry2);
        $taskG->query('COMMIT');
```

```
        if ($taskG->fetch()) {
            $id = $taskG->id;
}

        return $id;
```

Una vez solucionado el problema de conocer qué ID es el vinculado a la tarea creada, el siguiente paso ha sido crear la tarea para cada usuario del grupo.

Para ello, se ha tenido que crear una función que obtuviera los ids de los usuarios del grupo que no fueran profesores, ni administradores, ya que la tarea solamente se debe vincular a los alumnos.

Se ha creado por tanto la función de Grades *getMembersExcludeGradersAndAdmin*, con el siguiente contenido:

```
            $qry = 'select gm.profile_id as id '
                        . 'from group_member gm '
                        . 'where gm.is_admin <> 1 '
                        . 'and gm.group_id = ' . $groupid
                        . ' and gm.profile_id not in '
                        . '(select gg.userid '
                        . 'from grades_group gg '
                        . 'where gg.groupid = ' . $groupid . ')';
```

Una vez obtenidos los ids, solamente queda registrar para cada uno de ellos la nueva tarea. Para esto, se ha tenido que crear la función de Task, *register*.

La consulta se esta función también se ha creado de manera especial. Para evitar lanzar tantas consultas como alumnos haya, se ha optado por realizar una consulta de inserción múltiple, de manera que con una sola consulta se registre la tarea para todos los usuarios.

El código de esta consulta es el siguiente:

```
$qry = 'INSERT INTO task (id,userid,status) values ';
        for ($i = 0; $i < count($idsUsers); $i++) {
$qry .= '(' . $id . ',' . $idsUsers[$i] . ', 0)';
            if ($i < (count($idsUsers) - 1))
$qry .= ', ';
}
```

Todo esto sería la parte correspondiente a crear una tarea no existente. Si por el contrario el estado de la tarea fuera distinto de -1, quiere decir que la tarea ya existía, y por tanto el comportamiento es diferente.

```
Task_Grader::updateTask($taskid, $tag);
  Task::reopenTask($taskid);
```

```
  $result[0] = 1;
  $result[1] = $taskid;
```

En este caso, lo que se debe hacer es actualizar la tarea, para lo que se ha tenido que crear la función *updateTask*.

Esta función se va a encargar de colocar el estado de la tarea de nuevo a 1, y de actualizar el tag, en el caso de que se haya modificado.

```
if ($tag == "") {
            $qry = 'UPDATE task_grader ' .
                    'SET status=1 ' .
                    'WHERE id=' . $taskid;
        } else {
            $qry = 'UPDATE task_grader ' .
                    'SET status=1, tag="' . $tag . '" ' .
                    'WHERE id=' . $taskid;
        }
```

El siguiente paso es actualizar las tareas de los alumnos, para colocarlas de nuevo como pendientes. En este caso, y tal y como se explica en la tarea *Crear función CancelTask*, al cancelar una tarea, los registros vinculados a los alumnos no se eliminan, tan sólo se cambia su estado a 2 (cancelada). Por ello, para reabrir la tarea es tan sencillo como utilizar la función reopenTask, que lo único que hace es colocar el campo status de nuevo a 0.

Como los ID's de las tareas son únicos, con una sola consulta se puede cambiar el status de todos los alumnos.

```
$qry = 'UPDATE task ' .
                'SET status=0 ' .
'WHERE id=' . $taskid;
```

Hasta aquí el proceso de iniciar una tarea ya previamente creada. El siguiente paso, independientemente de que la tarea fuera nueva, o ya estuviera iniciada, es utilizar el esquema AJAX para evitar que se recargue la página, y sin embargo recargar el botón Iniciar para cambiar su estado a "iniciada", y deshabilitarlo.

```
if ($this->boolean('ajax')) {
            $this->startHTML('text/xml;charset=utf-8');
            $this->elementStart('head');
            // TRANS: Title.
            $this->element('title', null, _m('Add to favorites'));
            $this->elementEnd('head');
            $this->elementStart('body');
            $form = new InitForm($this, $groupid, $result);
            $form->show();
            $this->elementEnd('body');
```

```
$this->elementEnd('html');
        }
```

Crear histórico de tareas

Ficheros modificados: TaskcreateAction.php, Task_Grader.php

En la vista de profesor, cada grupo va a tener una lista con el histórico de sus tareas.

Para conseguir este funcionamiento, lo primero que se ha tenido que hacer es diseñar este apartado, y conocer el aspecto que se desea conseguir. Finalmente se ha decidido colocar un apartado de tareas debajo de cada grupo. En cada tarea, se ha optado por incluir información sobre su estado, la fecha en que se creó, el tag asociado, en caso de que tenga, y finalmente el número de alumnos que la han completado.

Una vez decidido esto, la implementación se ha llevado a cabo de la siguiente manera.

Lo primero que se ha hecho es implementar una función que devuelva a través de una sóla consulta toda la información que se va a necesitar. Se ha desarrollado así *getHistorical*, de Task_Grader, y su consulta es la siguiente:

```
$qry = 'select ' .
'(select count(id) from task where id = tg.id and status = 1) as completed, ' .
'(select count(id) as total from task where id = tg.id) as total, ' .
'tg.cdate as cdate, ' .
'tg.status as status, ' .
'tg.id as id, ' .
'tg.tag as tag ' .
'from task_grader tg ' .
'where tg.graderid = ' . $graderid .
' and tg.groupid = ' . $groupid .
' order by tg.cdate desc';
```

Una vez obtenida toda esta información en la acción TaskCreateAction, se ha creado un bloque div que alojará todo el histórico, y se le ha asociado una función JavaScript que se encargará de mostrarlo y ocultarlo.

Finalmente, para cada tarea, se ha incluido un nuevo bloque en el que se ha añadido la información devuelta por la consulta anterior, así como los formularios ReopenForm o CancelForm, dependiendo del estado de la tarea.

Una vez conseguido el histórico, se ha pasado a mejorar aún más su comportamiento a través de AJAX. De esta manera, cada vez que un profesor inicie una tarea, automáticamente se refrescará el histórico sin necesidad de recargar la página.

Para ello, el formulario de inicio de tarea, InitForm, tiene vinculado en el botón submit la función JavaScript *updateHistorical*. De esta manera, cuando el profesor pulse el botón enviar, se ejecutará el código de esta función.

Esta función utiliza la función load de JQuery para realizar una llamada a través de AJAX. Esta llamada tiene como destino un script que se ha tenido que desarrollar, updateHistorical.php.

Sin embargo, hay que mencionar un detalle importante. Como el evento JavaScript por lo general se ejecuta nada más accionar el evento onclick, aunque en la llamada se obtuvieran de nuevo las tareas, no se habría insertado la nueva, por lo que no se actualizaría nada. Para solucionar esto, y tras buscar opciones, se ha utilizado la función submit de JQuery, junto con *preventDefault*, consiguiendo que el código de la función submit de JavaScript se ejecute al finalizar la acción del formulario.

De esta manera, como la función load se encarga de recargar el contenido del elemento seleccionado, y ya que la función se ejecuta al finalizar el registro de la nueva tarea, en el script lo que se ha hecho es obtener de nuevo todo el histórico con la función getHistorical, de manera que ahora ya estará incluida la nueva tarea, y finalmente se han mostrado las tareas de la misma manera que en la acción, dando como resultado el mismo histórico, pero actualizado.

El código de la función JavaScript es el siguiente:

```
$('#init-task-group-' + groupid).submit(function(e) {
        e.preventDefault(); // don't submit multiple times
        setTimeout(function() {
            if ($('#div-group-task-' + groupid + ' #historical-' + groupid).css('display') === 'none') {
                $('#div-group-task-' + groupid + ' #historical-' + groupid).load("../local/plugins/Task/scripts/updateHistorical.php", {graderid: graderid, groupid: groupid});
            }
            else {
                $('#div-group-task-' + groupid + ' #historical-' + groupid).fadeOut("fast").load("../local/plugins/Task/scripts/updateHistorical.php", {graderid: graderid, groupid: groupid}).fadeIn('slow');
}
        }, 100);
    });
```

Crear función showConfirmDialog

Ficheros modificados: task.js

En la parte de los alumnos, cuando un usuario pulsa el botón Rechazar de una tarea, automáticamente se ejecuta la función de JavaScript showConfirmDialog para mostrar el cuadro de confirmación de rechazo.

Esta función JavaScript recibe el *taskid*, para poder realizar acciones sobre esa tarea desde JavaScript.

Lo que se hace en la función es seleccionar el cuadro de rechazo previamente creado en la acción, y aplicarle la función de JQuery *toogle*, que se encarga de mostrar u ocultar un elemento dependiendo de si está o no visible. En este caso, mostrará el cuadro de rechazo, y a la vez ocultará el bloque con las opciones “Completar” y “Rechazar”, utilizando de nuevo la función *toogle*.

```
function showConfirmDialog(id) {
    $("#div-task-" + id + ">#confirm-reject-task-" + id).toggle('fade', 300);
```

```
    $("#div-task-" + id + ">#options-task-" + id).toggle('fade', 300);
}
```

Crear función mostrarBox

Ficheros modificados: task.js

Cuando un alumno pulse el botón "Completar" relativo a una tarea, automáticamente se mostrará el cuadro de anotación previamente creado en la acción.

Para hacer esto se ha implementado la función JavaScript *mostrarBox*, la cual recibe el *taskid* para poder realizar acciones sobre la tarea desde JavaScript.

Posteriormente, lo que se hace es, por un lado, mostrar el cuadro de anotación utilizando la función de JQuery *toogle*. A su vez, se oculta el bloque con las opciones "Rechazar" y "Completar", utilizando la misma función.

```
$("#div-task-" + id + ">.input_form").toggle('fade', 300);
    $("#div-task-" + id + ">#options-task-" + id).toggle('fade', 300);
```

Por último, para localizar el foco en el cuadro de texto, y para colocar el cursor al final del mismo, ya que éste tiene contenido, se ha utilizado el siguiente código:

```
$("#div-task-" + id + " .notice_data-text").focus();
$("#div-task-" + id + " .notice_data-text").val($("#div-task-" + id + "
.notice_data-text").val() + ' ');
```

Rechazar una tarea

Ficheros modificados: TaskcreateAction.php, task.js, Task.php

Un alumno tiene la posibilidad de rechazar una tarea, para lo que deberá pulsar el botón "Rechazar", y posteriormente aceptar el cuadro de confirmación.

Si esto sucede, se ejecutará por un lado acción *taskcreate*, encargada de eliminar la tarea, y por otro lado el evento onclick deleteTask, quien se encargará de hacerla visualmente desaparecer, y de disminuir automáticamente el número de tareas pendientes.

En cuanto a la acción *taskcreate*, como al pulsar en el botón "Sí" del formulario de confirmación se envía un elemento *hidden* con el ID de la tarea, y con el valor *delete*, en la función *handle* de la acción ha sido posible conocer cuándo se procedía de este formulario.

```
else if ($this->trimmed('delete')) {
```

```
                $taskid = $this->trimmed('delete');
                $this->deleteTask($taskid);
```

Una vez dentro de la comprobación, únicamente se obtiene el valor del elemento hidden delete, que es el taskid, y se le pasa a la función deleteTask.

Para rechazar una tarea, internamente se traduce en modificar el estado de la tarea asociada al alumno a un -1. Para ello, se ha creado la función en Task, *rejectTask*, con el siguiente código:

```
$qry = 'UPDATE task ' .
                'SET status=-1 ' .
                'WHERE userid=' . $userid .
                ' AND id=' . $taskid;
```

Una vez creada esa función, desde la funcióndeleteTask únicamente se debe realizar una llamada a *rejectTask*, y finalmente utilizar el esquema Ajax para mantenerse en la página sin que ésta se recargue. El contenido del esquema Ajax no tiene importancia, ya que posteriormente, con la función JavaScript deleteTask se eliminará visualmente la tarea.

El funcionamiento de la función JavaScript es el siguiente:

Por un lado utiliza la función *toogle* junto con el parámetro *fade* para ocultar de forma progresiva la tarea. Po otro lado, se ha modificado el valor del número de tareas pendientes restándole uno, y se ha realizado un desvanecido.

```
$("#div-task-" + id).toggle('fade', 300);
    $(".pending-tasks-number").text($(".pending-tasks-number").text() - 1);
    $(".pending-tasks-number").css('display', 'none');
    $(".pending-tasks-number").toggle('fade', 800);
```

Con todos estos pasos, cuando un usuario rechace una tarea, automáticamente su estado se convertirá en -1, la tarea desaparecerá visualmente, y por otro lado el número de tareas pendientes disminuirá en 1.

Crear formulario para cancelar una tarea

Ficheros modificados: CancelForm.php, TaskPlugin. Php, task.css, task.js

Una tarea iniciada puede ser cancelada siempre y cuando nadie la haya completado aún. Para cancelarla es necesario disponer de un botón, en este caso este formulario, que permita realizar la acción asociada, *taskcreate*.

Este formulario se ha implementado en el fichero *CancelForm.php*.

Como la acción de cancelar se realiza sobre una tarea ya existente, tan sólo es necesario pasarle como parámetro el *taskid*, que es lo que necesitará la acción posterior para cancelar una tarea existente.

En cuanto a elementos, solamente se ha añadido un elemento *hidden*, a través del cual pasar el ID de la tarea, y por otro lado el botón de tipo submit para enviar el formulario y llamar a la acción asociada.

```
$this->out->hidden('cancel-task-h' . $this->taskid, $this->taskid, 'cancel-
task');
$this->out->element('input', array('type' => 'submit',
'value' => 'Cancelar', 'class' => 'cancel-task-button',
'title' => 'Cancela esta tarea'),
'onclick' => 'updateTaskStatus(' . $this->taskid . ');')););
```

Como se puede ver, se ha incluido un evento onClick para actualizar el estado de la tarea.
La función JavaScript asociada realiza lo siguiente:

```
$('#task-' + taskid + ' span:first').fadeOut().text($('#task-' + taskid + '
span:first').text() == 'Iniciada' ? 'Cancelada' : 'Iniciada').fadeIn();
```

Cancelar una tarea

Ficheros modificados: TaskcreateAction.php, Task.php, Task_Grader.php

A través del histórico de tareas, un profesor puede cancelar una tarea siempre y cuando no haya sido contestada por ningún alumno.

Para cancelarla deberá pulsar en el formulario *CancelForm.php*, y automáticamente se ejecutará la acción asociada, *taskcreate*.

En la función *handle* se comprobará si el elemento *hidden* pasado a través del formulario se llama *cancel-task*, y de ser así, se ejecutará la función *CancelTaskGrader*.

```
else if ($this->trimmed('cancel-task')) {
                $taskid = $this->trimmed('cancel-task');
$this->cancelTaskGrader($taskid);
}
```

Para cancelar una tarea, es necesario colocar el estado de la tarea a 0, y actualizar el estado de las tareas de los usuarios a 2 (cancelada).

Para ello, se han creado las funciones *Task_Grader::cancel*, y *Task::cancelTask*.

Una vez creadas, desde CancelTaskGrader únicamente hay que llamar a ambas, y utilizar el esquema Ajax para recargar en el formulario pulsado el formulario ReopenForm.

Reabrir una tarea

Ficheros modificados: TaskcreateAction.php, Task.php, Task_Grader.php

Desde el histórico de tareas, un profesor puede decidir reabrir una tarea previamente cancelada. El proceso es muy parecido al de cancelar la tarea.

Al pulsar en el formulario *ReopenForm.php*, se ejecutará la acción asociada, en este caso, *taskcreate*.

En la función *handle* de esta acción se comprobará si el elemento *hidden* se llama reopen-task, y de ser así, se llamará a la función ReopenTask.

Para reabrir una tarea hay que cambiar el estado de la tarea del grader a 1 (iniciada), y de la tarea de los alumnos a 0 (pendiente). Para ello se han creado las funciones *Task_Grader::reopenTask* y *Task::reopenTask.*

Una vez creadas las funciones, únicamente se debe llamar a ambas, y utilizar el esquema Ajax para cargar en el formulario pulsado el formulario Cancelform.php.

Crear formulario para reabrir una tarea

Ficheros modificados: TaskcreateAction.php, ReopenForm.php, task.css, task.js

Cuando una tarea ha sido cancelada, es posible reabrirla cuando se considere oportuno.

En este caso el botón que lo permite se ha implementado a través del formulario ReopenForm, en el fichero *ReopenForm.php*.

Como la tarea ya ha sido creada, la única información que se necesita para acceder a ella en la base de datos y modificar su estado es el ID, por lo que el constructor del formulario tan sólo necesita como parámetro adicional el *taskid*.

En cuanto a elementos, se ha incluido un elemento hidden para el ID de la tarea, y por otro lado un botón de tipo submit para enviar el formulario y llamar a la acción asociada, *taskcreate*.

```
        $this->out->hidden('reopen-task-h' . $this->taskid, $this->taskid, 'reopen-
        task');
        $this->out->element('input', array('type' => 'submit',
        'class' => 'reopen-task-button',
        'value' => 'Iniciar',
'title' => 'Reabre esta tarea.') ,
'onclick' => 'updateTaskStatus(' . $this->taskid . ');'));
```

De la misma manera que en el formulario para cancelar una tarea, se ha vinculado a través del evento onclick la función *updateTaskStatus*, para actualizar el estado de la tarea.

Crear función hideConfirmDialog

Ficheros modificados: task.js

Cuando un usuario pulsa "Rechazar" en una tarea, automáticamente se muestra un cuadro de confirmación. Este cuadro de confirmación ofrece un botón con una "X" para cerrarlo, así como un botón "No".

Cualquiera de ellos tiene vinculada la función JavaScript *hideConfirmDialog*. Esta función se encarga de ocultar el cuadro de confirmación, y de mostrar de nuevo el bloque con los botones de "Rechazar" y "Completar", utilizando la función de JQuery *toogle*.

```
$("#div-task-" + id + ">#confirm-reject-task-" + id).toggle('fade', 300);
    $("#div-task-" + id + ">#options-task-" + id).toggle('fade', 300);
```

Crear acción para recibir el envío de una tarea

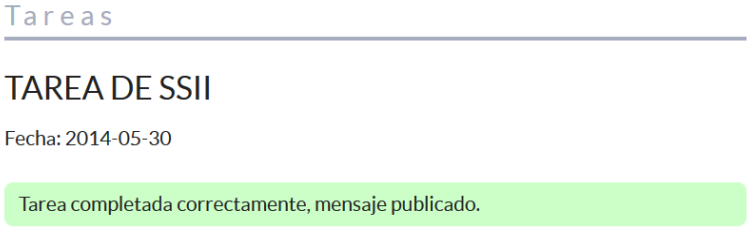
Ficheros modificados: TaskPlugin. Php, NewnoticetaskAction.php, Task.php, task.css

Cuando un usuario envía una tarea, es necesario validarla y realizar algunas comprobaciones antes de almacenarla en la base de datos.

Todo esto se ha realizado en la acción asociada al formulario NoticeTaskForm, *NewnoticetaskAction*.

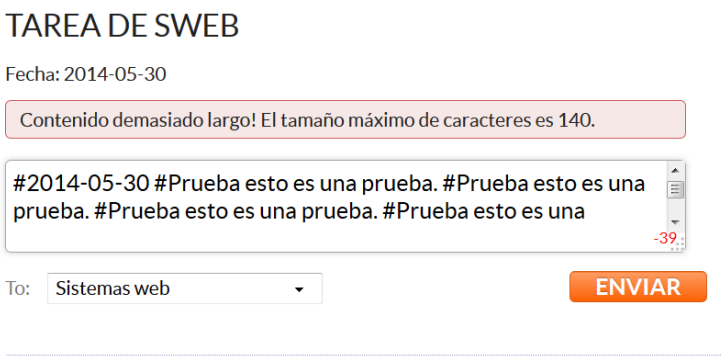
Esta acción está basada en la acción original *newnotice.php*. En esta nueva acción, se han realizado algunos aspectos diferentes. Se ha tenido que adaptar la comprobación al nuevo formulario, ya que en este se habían eliminado algunos campos originales.

Si todas las comprobaciones referentes a la anotación son válidas, desde esta acción se llamará a la función *Task::completeTask*, que lo que hará será colocar el estado de la tarea a 1 (completada), y posteriormente con el esquema AJAX se sustituirá el formulario por un mensaje de información.



Si alguna comprobación produce un error, se ha capturado a través de un catch, y lo que se ha hecho es aprovechar el esquema AJAX, y el mecanismo de errores introducido en el formulario NoticeTaskForm, para poder generar el mismo formulario, con el mismo contenido, pero con un mensaje de error.

Ya que en el esquema AJAX no es posible introducir ningún script, y por tanto no era posible añadir un mensaje de error al formulario, la solución ha sido añadir un mecanismo de errores al formulario, de manera que en caso de error, lo que se hace es crear de nuevo el formulario, pero esta vez utilizando el último parámetro *msg* para mostrar el error.



Para no perder el contenido al crear de nuevo el formulario, se ha obtenido el contenido actual de la caja al iniciar la acción, y posteriormente se le ha pasado al constructor del formulario.

```
$this->content = $this->trimmed('status_textarea');
```

…

```
$notice_form = new NoticeTaskForm($this, $taskid, array('content' => $this->content, 'to_group' => $group), $msg);
$notice_form->show();
```

Crear contador de caracteres

Ficheros modificados: task.js, NoticeTaskForm.php

Al crear un formulario para escribir las tareas diferente al por defecto, el contador interno de StatusNet no se aplicaba a este nuevo formulario.

Para solucionar esto, se ha tenido que implementar un contador manualmente. Para ello, lo primero ha sido asociar la función *counter* de JavaScript al elemento text_area en la creación del formulario NoticeTaskForm, a través del evento onfocus.

```
'onfocus' => 'counter(' . $this->taskid . ');'
```

De esta manera, cada vez que este elemento obtenga el foco se ejecutará la función *counter*. Por otro lado, esta función lo que realiza es un control de caracteres a medida que el usuario va escribiendo en el text_area.

Para conseguir esto, se ha realizado de la siguiente manera:

```
limit = 140;
    $("#div-task-" + id + " .notice_data-text").bind('input propertychange', function() {
        if ((limit - $(this).val().length) < 0) {
            $("#div-task-" + id + " .count").addClass('count-negative');
        }
        else {
            $("#div-task-" + id + " .count").removeClass('count-negative');
        }
```

```
  $("#div-task-" + id + " .count").text(limit - $(this).val().length);
});
```

Como se puede ver, se asocia el evento input propertychange a la caja de texto. Este evento está controlado por la función anónima que se muestra en el código. En esta función lo que se hace en cada modificación del text_area es actualizar el número de caracteres restantes, y comprobar si el número obtenido es mayor o menor que 0, de manera que se le aplique la clase count-negative, que tiene unas reglas asociadas para mostrar el contador en color rojo.

Disminuir el número de tareas pendientes al enviar una tarea

Ficheros modificados: task.js, NoticeTaskForm.php

La idea es que cuando un usuario envíe su tarea, automáticamente se refresque el número de tareas pendientes. Sin embargo, el problema es que esto se podría hacer directamente con JavaScript, pero al ejecutarse antes de la llamada a la acción vinculada con el botón enviar, es posible que el número de tareas se disminuya y luego la acción produzca un error, por ejemplo de exceso de tamaño, y sin embargo el número se haya disminuido.

Para evitar esto, y utilizando el mismo comportamiento que al actualizar el histórico de tareas, lo que se ha hecho es utilizar la función submit de JavaScript junto con *preventDefault*, para conseguir que el código JavaScript se ejecute al finalizar la acción del formulario.

Sin embargo, hay que controlar que el submit se haya realizado correctamente, por lo que se ha tenido que realizar una comprobación.

Si todo se ha ejecutado correctamente, entonces sí se disminuye mediante JavaScript el número de tareas pendientes.

El código de la función es el siguiente:

```
$('#form_notice_task_' + taskid).submit(function(e) {
        e.preventDefault(); // don't submit multiple times
        setTimeout(function() {

            if ($('#form_notice_task_' + taskid).length == 0) {
                $(".pending-tasks-number").text($(".pending-tasks-number").text()  -
1);
                $(".pending-tasks-number").css('display', 'none');
                $(".pending-tasks-number").toggle('fade', 800);
}

        }, 1000);
    });
```

Para que se ejecute este código JavaScript, en el botón submit del formulario NoticeTaskForm se ha añadido un evento onclick vinculado a esta función:

```
'onclick' => 'reducir('.$this->taskid.');',
```

9 Bibliografía

1. Scott, P. "The Meanings of Mass Higher Education" Open University Press (1995).

2. Reichert, S. Tauch,C.: "Bologna four years after: Steps towards sustainable reform of higher education in Europe" European University Association (2003)

3. Bonk, C.J., Kim. A. "Extending sociocultural theory to adult learning" In: Smith, M.C., Pourchot, T. (eds) Adult learning and development: Perspectives from educational psychology, pp. 67-88 Lawrence Erlbaum Associates (1998).

4. Johnson, D. W., Johnson, R. T., Smith, K. A. "Cooperative learning: Increasing college faculty instructional productivity" ASHE-ERIC Higher Education Report No.4 School of Education and Human Development, The George Washington University (1991).

5. Rosenberg, M. J.: E-learning "Strategies for Delivering Knowledge in the Digital Age". McGraw-Hill (2001).

6. Michaelsen, L.K., Black, R. H. "Building learning teams: The key to harnessing the power of small groups in higher education". In: Kadel, S., Keehner, J, (eds.) Collaborative Learning: A Sourcebook for Higher Education, vol. 2, pp. 65-81. National Center for Teaching, Learning and Assessment (1994).

7. McKeachie, W.J. "Teaching Tips, Strategies, Research and Theory for College and University Teachers", 9ª ed., Lexington. Heath and Co (2002).

8. Mason, R., Rennie, F.: E-learning and Social Networking Handbook: Resources for Higher Education, Routledge (2008).

9. Zhao, D., Rosson, M.B. "How and why people Twitter: the role that micro-blogging plays in informal communication at work". In: 2009 ACM International Conference on Supporting Group Work (GROUP 2009), pp. 243-252. ACM Press (2009).

10. Alexa, "Alexa Traffic ranking," 2010. Retrieved on June 15, 2010 from: http://www.alexa.com/

11. Sancho-Thomas, P., Fuentes-Fernández, R., Fernández-Manjón, B.: Learning teamwork skills in university programming courses. Computers and Education 53(2), pp. 517-531.

12. Livinstone, B. "Using Web 2.0 Technologies". ASTD Press (2010).

13. Grosseck, G., Holotescu, C. "Can we use twitter for educational activities?" In: 4th International Scientific Conference eLearning and Software for Education (eLSE 2008), (2008).

14. Lowenthal, P.R., Thomas, D., Thai, A., Tuhnke, B. “The CU Online Handbook”. University of Colorado Denver (2009)

15. Java, A., Song, X., Finin, T., and Tseng, B., “Why We Twitter Understanding Microblogging Usage and Communities”, In: Procedings of the Joint 9th WEBKDD and 1st SNA-KDD Workshop (2007)

16. Ebner, M., Schiefner, M., “Microblogging - more than fun?” In: Proceedings of IADIS Mobile Learning Conference 2008, Inmaculada Arnedillo Sánchez and Pedro Isaías ed., Portugal, p. 155-159 (2008)

17. Milstein, S., Lorica, B.,”Twitter and the Micro-Messaging Revolution: Communication, Connections, and Immediacy—140 Characters at a Time”, O’Reilly Media (2008).

18. Boyd, D., Ellison, N. ,“Social Network Sites: Definition, History, and Scholarship”, in: Journal of Computer-Mediated Communication 13/2008, 210–230.

19. “The Bologna Process 2020 - The European Higher Education Area in the new decade”, http://www.eees.es/pdf/Leuven_Louvain-la-Neuve_Communique_April_2009.pdf.

20. Aspden, E.J. & Thorpe, L.P.. "Where Do You Learn?: Tweeting to Inform Learning Space Development”. *Educause Quarterly, 32*(1), (2009)

21. Perez, E., “Professors experiment with Twitter as teaching tool”, in: Journal Sentinel, (2009)

22. Ebner, M., Lienhardt, C., Rohs, M., Meyer, I., “Microblogs in High Education – A chance to facilitate informal and process-oriented leaning?” In: Computers & Education, 55/2010, 92-100.

23. Miners , Z. “Twitter Takes a Trip to College” in: U.S. News & World, 146/2009, 56-57.

24. Rejon-Guardia, F., Sanchez-Fernandez, J. & Munoz-Leiva, F. “The acceptance of microblogging in the learning process: the μBAM model” Journal of Technology and Science Education (JOTSE), 3(1), 31-47.

25. Gómez-Sanz, J. , Fuentes-Fernández, Rubén , Arroyo, Javier , Blanco, Diego , Gutiérrez-Cosío, Celia, Pavón, Juan “Experiences on the Application of Micro-Blogging to Promote Learning Autonomy in University Courses”, Actas del ISELEAR 2011.

26. Gómez-Sanz, J., Fuentes, Rubén, Gutierrez Cosio, Celia, and Pavon, J. "Using microblogging tools for tracking daily activities of students in university courses." *Talleres de las Jornadas de Ingeniería del Software y Bases de Datos*, Actas ISELEAR 2010.

27. Gomez Sanz, Jorge, Gutierrez Cosio, Celia, Fuentes, Rubén and Pavon, Juan "Microblogging in the European Higher Education Area." Actas *1st International Conference on European Transnational Education, ICEUTE 2010*.

28. Instituto Nacional de Estadística, Encuesta sobre Equipamiento y Uso de Tecnologías de Información y Comunicación en los hogares, 2013.

29. *Comscore, Spain Digital Future in Focus, 2013 (http://www.comscore.com/).*

30. Cole, J., & Foster, H. "*Using Moodle: Teaching with the popular open source course management system"*. O'Reilly Media, Inc. (2008)

31. Tsur, O., & Rappoport, A. "What's in a hashtag?: content based prediction of the spread of ideas in microblogging communities". In *Proceedings of the fifth ACM international conference on Web search and data mining* (pp. 643-652). ACM. (2012)

32. Grosseck, G., & Holotescu, C." Microblogging multimedia-based teaching methods best practices with Cirip. Eu". *Procedia-Social and Behavioral Sciences*, *2*(2), 2151-2155. (2010).

33. Twitter Usage, 2014. https://about.twitter.com/company

34. StatusNet Wiki, 2012, http://status.net/wiki/Main_Page

35. CLOC, 2013, http://cloc.sourceforge.net/

36. GitStats, 2013, http://sourceforge.net/projects/gitstats/